UNIVERSITY OF CALIFORNIA

Los Angeles

The Material Point Method for the Physics-Based Simulation of Solids and Fluids

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Chenfanfu Jiang

2015

ABSTRACT OF THE DISSERTATION

The Material Point Method for the Physics-Based Simulation of Solids and Fluids

by

Chenfanfu Jiang

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2015

Professor Demetri Terzopoulos, Co-chair

Professor Joseph M. Teran, Co-chair

Simulating fluids and solid materials undergoing large deformation remains an important and challenging problem in Computer Graphics. The dynamics of these materials usually involve dramatic topological changes and therefore require sophisticated numerical approaches to achieve sufficient accuracy and visual realism. This dissertation focuses on the Material Point Method (MPM) for simulating solids and fluids for use in computer animation, and it makes four major contributions:

First, we introduce new MPM for simulating viscoelastic fluids, foams and sponges. We design our discretization from the upper convected derivative terms in the evolution of the left Cauchy-Green elastic strain tensor. We combine this with an Oldroyd-B model for plastic flow in a complex viscoelastic fluid. While the Oldroyd-B model is traditionally used for viscoelastic fluids, we show that its interpretation as a plastic flow naturally allows us to simulate a wide range of complex material behaviors. In order to do this, we provide a modification to the traditional Oldroyd-B model that guarantees volume preserving plastic flows. Our plasticity model is remarkably simple (foregoing the need for the singular value decomposition (SVD) of stresses or strains). We show that implicit time stepping can be achieved in a manner that enables high resolution

simulations at practical simulation times.

Particle-in-Cell (PIC) techniques, particularly the Fluid Implicit Particle (FLIP) variants have become standard in computer graphics. While they have proven very powerful, they do suffer from some well known limitations. Our second contribution is to introduce a novel technique designed to retain the stability of the original PIC, without suffering from the noise and instability of FLIP. Dissipation in the original PIC results from a loss of information when transferring between grid and particle representations. We prevent this loss of information by augmenting each particle with a locally affine, rather than locally constant, description of the velocity. We show that this not only stably removes the dissipation of PIC, but that it also allows for exact conservation of angular momentum across the transfers between particles and the grid.

Our third contribution is to introduce a novel material point method for heat transport, melting and solidifying materials. This brings a wider range of material behaviors into reach of the already versatile MPM. Extending the material point method in this way requires several technical novelties. We introduce a dilational/deviatoric splitting of the constitutive model and show that an implicit treatment of the Eulerian evolution of the dilational part can be used to simulate arbitrarily incompressible materials. Furthermore, we show that this treatment reduces to a parabolic equation for moderate compressibility and an elliptic, Chorin-style projection at the incompressible limit. Since projections are naturally done on marker and cell (MAC) grids, we devise a staggered grid MPM method. To generate varying material parameters, we adapt a heat-equation solver to the material point framework.

Practical time steps in state-of-the-art simulators typically rely on Newton's method to solve large systems of nonlinear equations. In practice, this works well for small time steps but is unreliable at large time steps at or near the frame rate, particularly for difficult or stiff simulations. Our fourth contribution is to show that recasting the backward Euler method as a minimization problem allows Newton's method to be stabilized by

standard optimization techniques with some novel improvements of our own. The resulting solver is capable of solving even the toughest simulations at the 24 Hz frame rate and beyond. We show how simple collisions can be incorporated directly into the solver through constrained minimization without sacrificing efficiency. We also present novel penalty collision formulations for self collisions and collisions against scripted bodies designed for the unique demands of this solver. We show that these techniques improve the behavior of MPM simulations.

The dissertation of Chenfanfu Jiang is approved.

Song-Chun Zhu

Stanley Osher

Joseph M. Teran, Committee Co-chair

Demetri Terzopoulos, Committee Co-chair

University of California, Los Angeles

2015

To my family and friends

Table of Contents

1 Introduction **1**

1.1 Contributions 2

1.2 Dissertation Overview 3

2 The Material Point Method **6**

2.1 Deformation 6

2.1.1 Useful differentials 6

2.2 The MPM algorithm 10

2.3 Snow simulation results 12

3 Elasticity and Plasticity **14**

3.1 Hyperelasticity 14

3.1.1 Mass-spring system 15

3.1.2 Neo-Hookean model 15

3.1.3 Fixed corotated model 16

3.2 Plasticity 17

3.2.1 Plastic flow 17

3.2.2 Material hardening 19

3.3 Lagrangian forces 19

4 Simulating Viscoelastic Fluids, Foams and Sponges **23**

4.1 Background 23

4.2 Governing equations 25

4.2.1 Upper convected derivative 26

4.2.2 Volume-preserving plasticity 27

4.2.3 Elasticity 28

4.3 Discretization 28

4.4 Simulation results . . . 30
4.5 Discussion . . . 35

5 The Affine Particle-in-Cell Method . . . 36
5.1 Background . . . 37
5.2 Method Outline . . . 41
5.3 Particle-grid transfers . . . 42
5.3.1 PIC . . . 42
5.3.2 Rigid Particle-In-Cell (RPIC) . . . 43
5.3.3 Affine Particle-In-Cell (APIC) . . . 45
5.4 Fluids . . . 47
5.5 Simulation results . . . 47
5.6 Discussion . . . 54

6 Simulating Melting and Solidification . . . 56
6.1 Background . . . 56
6.2 Method Overview . . . 60
6.3 Physical Model . . . 64
6.3.1 Heat flow and phase transition . . . 64
6.3.2 Constitutive model . . . 66
6.3.3 Pressure Splitting . . . 68
6.3.4 Pressure . . . 68
6.3.5 Temporal evolution . . . 69
6.3.6 Discretization . . . 69
6.4 Algorithm . . . 70
6.4.1 Apply plasticity from previous timestep . . . 72
6.4.2 Compute interpolation weights . . . 72
6.4.3 Rasterize particle data to grid . . . 75
6.4.4 Classify cells . . . 76

6.4.5 MPM velocity update . . . 76
6.4.6 Process the grid collisions . . . 78
6.4.7 Project the velocities . . . 79
6.4.8 Solve the heat equation . . . 80
6.4.9 Update the particle state from the grid . . . 80
6.4.10 Process the particle collisions and positions . . . 82
6.5 Simulation results . . . 82
6.6 Discussion . . . 88
6.7 Summary . . . 90

7 Conclusion . . . 91

A An Optimization-Based Integrator . . . 94
A.1 Introduction . . . 94
A.2 Time Integration . . . 96
A.2.1 Minimization problem . . . 99
A.3 Minimization . . . 101
A.3.1 Unconstrained minimization . . . 101
A.3.2 Constrained minimization . . . 104
A.3.3 Practical considerations . . . 106
A.4 Forces . . . 107
A.4.1 Elastic . . . 107
A.4.2 Damping . . . 108
A.5 Collisions . . . 109
A.5.1 Object collisions as constraints . . . 110
A.5.2 Object penalty collisions . . . 111
A.5.3 Penalty self-collisions . . . 112
A.6 Accelerating the MPM . . . 113
A.6.1 Optimization formulation . . . 114

A.6.2 Particle position update . . . 115
A.7 Simulation results . . . 116
A.8 Summary . . . 120

B Derivatives for the Oldroyd-B Model . . . 123

C RPIC and APIC Proofs . . . 126
C.1 Preliminaries . . . 126
C.2 Piecewise rigid . . . 127
C.2.1 Preservation of rigid motion . . . 127
C.2.2 Conservation of momentum . . . 129
C.2.3 Conservation of angular momentum . . . 130
C.3 Affine . . . 131
C.3.1 Preservation of affine velocity fields . . . 132
C.3.2 Conservation of momentum . . . 133
C.3.3 Conservation of angular momentum . . . 134

D Derivatives for Deviatoric Elasticity . . . 136

Bibliography . . . 138

List of Figures

2.1 A snowball smashes into a wall and sticks to it 12
2.2 A snowball drops to the ground 13
2.3 Two snowballs smash into each other 13

3.1 An APIC coupled simulation of yogurt and cloth 21
3.2 Comparing APIC with FLIP and PIC using the Lagrangian force model 21

4.1 Twisting sponge 30
4.2 Shooting at a sponge 30
4.3 A simulation of shaving foam 31
4.4 A simulation of toothpaste 32
4.5 Viennetta ice cream 33
4.6 Throwing a pie at a mannequin 34

5.1 Performance comparison with some simple 2D examples 39
5.2 The basic dataflow of various hybrid particle/grid simulation techniques 41
5.3 APIC simulation of a rushing river 48
5.4 Comparing the methods with a fountain simulation of free-surface flow 48
5.5 APIC vs FLIP in a wine pouring example 49
5.6 APIC vs FLIP in an MPM simulation 50
5.7 APIC vs FLIP in a granular material example 51
5.8 APIC vs FLIP in a high energy collision example 52
5.9 The performance of APIC with an elastoplastic model for lava flow . . . 53
5.10 Comparing APIC with FLIP and PIC during a lava in free-fall example . 53
5.11 Relative timing 54

6.1 Interplay of grids and particles 63
6.2 Cell classification criteria and stencils for a particle 73
6.3 Effect of density correction on stationary pool simulation 80

6.4 Mixtures of materials . . . 83
6.5 Bringing a hot fluid stream in contact with a cold solid . . . 84
6.6 Melting objects from the outside . . . 85
6.7 An apple is pulled from liquid candy . . . 86
6.8 Melting butter . . . 86
6.9 Changing the value of latent heat affects the rate of phase transition . . . 87
6.10 Lava solidifying into pāhoehoe . . . 87

A.1 Convergence plot of Newton's method and our method . . . 97
A.2 Cube being stretched and then given a small compressive pulse . . . 98
A.3 Cube being stretched . . . 98
A.4 Two spheres fall and collide with one another . . . 99
A.5 Line search . . . 105
A.6 Sphere dropping hard on the ground . . . 109
A.7 Random displaced particles test . . . 117
A.8 Point test with clapsed particles . . . 117
A.9 A torus falls on the ground . . . 117
A.10 A torus is pushed through a hole . . . 118
A.11 Stack of boxes . . . 118
A.12 An armadillo is squeezed between 32 rigid cubes . . . 119
A.13 125 tori are dropped into a bowl . . . 119

List of Tables

2.1 Performance comparison with the original formulation 13

4.1 Material parameters 32
4.2 Simulation performance 34

6.1 Quantities stored on each particle. 71
6.2 Resolutions and simulation times 87

A.1 Timing info 121

ACKNOWLEDGMENTS

I wish to express my sincere thanks to my advisors, Professor Demetri Terzopoulos and Professor Joseph Teran, for their continuous support of my PhD study and research. Their insight and expertise greatly helped me grow as a researcher. Without their guidance and encouragement this thesis would never have materialized. I also thank them for providing me many opportunities for my career.

I also thank Professor Stanley Osher and Professor Song-Chun Zhu for serving on my thesis committee and providing suggestions on improving and extending my work.

My appreciation goes to my UCLA colleagues for their help and insights. I would like to express my gratitude to Jingyi Fang, Xiaowei Ding, Sharath Gopal, Gergely Klar, Masaki Nakada, Eduardo Poyart, Garett Ridge, Matthew Wang, Tomer Weiss, Wenjia Huang, Kresimir Petrinec, Weiguang Si, Lap-Fai Yu, Andre Pradhana, Jan Hegemann, Theodore Gast, Daniel Ram, Craig Schroeder, Yuting Wang, Russell Howes, Jan Hegemann, Alexey Stomakhin and Jeffrey Hellrung.

I would like to thank the awesome people with whom I worked at Disney Animation Studios and Industrial Light & Magic. In particular, I would like to thank Andrew Selle, Lawrence Chai, Alexey Stomakhin, Aleka McAdams, Hao Li and Kiran Bhat for sharing their knowledge and guiding me on my work.

I thank my wife Chuchu for her persistent love. She has been a great companion as I have chased my dreams. She always makes me feel comfortable and confident whenever frustration and anxiety attack me.

Most of all, I would like to deeply thank my parents for their support and unconditional love. Their trust, patience and encouragement have enabled me to reach this far.

VITA

2010	B.S. (Physics), Special Class for the Gifted Young, University of Science and Technology of China.
2012–2013	Teaching Assistant, UCLA.
2011–2015	Research Assistant, UCLA.
2012	M.S. (Computer Science), UCLA.
Summer 2012	Research Intern, Industrial Light & Magic, Lucasfilm, San Francisco, CA.
Summer 2013	Research Intern, Walt Disney Animation Studios, Burbank, CA.
2014-2015	Consultant, Walt Disney Animation Studios, Burbank, CA.

PUBLICATIONS

J. Hegemann, C. Jiang, C. Schroeder, J. Teran, A Level Set Method for Ductile Fracture, *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 193-201, 2013. (Best Paper Award)

J. Fang, C. Jiang, D. Terzopoulos, Modeling and animating Myriapoda: A real-time kinematic/dynamic approach, *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 203-212, 2013.

Y. Wang, C. Jiang, C. Schroeder, J. Teran, An Adaptive Virtual Node Algorithm with Robust Mesh Cutting, *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pp. 77-85, 2014.

A. Stomakhin, C. Schroeder, C. Jiang, L. Chai, J. Teran, A. Selle, Augmented MPM for Phase-Change and Varied Materials, *ACM Transactions on Graphics (SIGGRAPH 2014 Proceedings)*, 33(4), 2014.

T. Gast, C. Schroeder, A. Stomakhin, C. Jiang, J. Teran, Optimization Integrator for Large Time Steps, *IEEE Transactions on Visualization and Computer Graphics*, 2015

C. Jiang, C. Schroeder, A. Selle, J. Teran, A. Stomakhin, The Affine Particle-In-Cell Method, *ACM Transactions on Graphics (SIGGRAPH 2015 Proceedings)*, 34(4), 2015.

D. Ram, T. Gast, C. Jiang, C. Schroeder, A. Stomakhin, J. Teran, P. Kavehpour, A Material Point Method for Viscoelastic Fluids, Foams and Sponges, *ACM SIGGRAPH/ Eurographics Symposium on Computer Animation (SCA)*, 2015.

CHAPTER 1

Introduction

Simulating natural phenomena for virtual worlds and characters is an important application in Computer Graphics that remains extremely challenging. An artist's need to manipulate and comprehend physical simulations imposes a significant constraint, all but requiring simulation methods to involve Lagrangian particles. While Lagrangian approaches are best for simulating solid-like behavior and Eulerian approaches most easily simulate fluid-like behavior, some materials are in the middle ground. These materials can exhibit elastic, solid-like resistance to deformation, but they can also undergo very large strains and complex topological changes characteristic of fluids. Discretization of such topology-changing materials is challenging because of the nonlinear governing equations and the wide range of exhibited behaviors.

The need for computational efficiency, topological variability, and numerical stability has led engineers toward hybrid, Lagrangian/Eulerian methods. Because of the use of a Cartesian grid, these methods are naturally suited to resolve topology changes and self-collisions combined with the Lagrangian tracking of mass, momentum and deformation on particles. In practice, the particle-wise deformation information can be used to represent quantities such as elastoplastic stresses arising from changes in shape, while an Eulerian background grid is used for implicit solves.

Particle-in-Cell (PIC) techniques, particularly the Fluid Implicit Particle (FLIP) variants have become the norm in computer graphics fluid simulation. The Material Point Method (MPM), an extension of FLIP that addresses solid mechanics, has recently

been introduced to the physics-based animation community, and successfully used to simulate elastoplastic flows to animate snow [Stomakhin et al., 2013], phase-changing materials [Stomakhin et al., 2014], Herschel-Bulkley plastic flows for foam [Yue et al., 2015], and some other materials [Jiang et al., 2015; Ram et al., 2015].

1.1 Contributions

To our knowledge, this is the first PhD dissertation on the MPM in the field of Computer Graphics. It makes the following key contributions:

- We introduce a novel information transfer scheme—the Affine Particle-In-Cell (APIC) method—for hybrid Lagrangian/Eulerian simulations. The method is stable and accurate without introducing complexity, and we apply it in various simulations (e.g., Figure 5.5 and Figure 5.9).

- We derive a deviatoric/dilational splitting technique for arbitrary constitutive models, which enables the unified simulation of compressible and incompressible materials (e.g., Figure 6.4) as well as transitions between them (e.g., Figure 6.6).

- We develop a heat equation solver that naturally couples with the MPM to easily simulate melting (e.g., Figure 6.5) and freezing (e.g., Figure 6.10) phenomena.

- We develop a modified, volume-preserving Oldroyd model for viscoelasticity and apply it in the framework of the MPM to simulating non-Newtonian materials, such as foams and sponges (e.g., Figure 4.1 and Figure 4.3).

- We present an optimization-based time integration solver that improves the stability and efficiency of both FEM (e.g., Figure A.7) and MPM (e.g., Figure 2.3) simulations, such as to simulate dynamic snow (e.g., Figure 2.1).

- We develop a coupling technique for simulating particle and mesh based materi-

als together (e.g., Figure 3.1).

1.2 Dissertation Overview

The remainder of this dissertation is structured as follows:

Chapter 2 and Chapter 3 briefly review the basic MPM algorithm, as well as the simplest constitutive models that are commonly used for elastic and plastic objects. We assume that readers have basic knowledge about continuum mechanics and the MPM. In particular, Chapter 3 covers the snow constitutive model introduced by Stomakhin et al. [2013].

Chapter 4 presents a constitutive model for simulating viscoelastic fluids, foams and sponges [Ram et al., 2015]. We design our discretization from the upper convected derivative terms in the evolution of the left Cauchy-Green elastic strain tensor. We combine this with an Oldroyd-B model for plastic flow in a complex viscoelastic fluid. While the Oldroyd-B model is traditionally used for viscoelastic fluids, we show that its interpretation as a plastic flow naturally allows us to simulate a wide range of complex material behaviors. To this end, we provide a modification to the traditional Oldroyd-B model that guarantees volume-preserving plastic flows. Our plasticity model is remarkably simple (foregoing the need for the singular value decomposition (SVD) of stresses or strains).

Chapter 5 presents a new hybrid, Lagrangian/Eulerian method, the Affine Particle-In-Cell (APIC) method [Jiang et al., 2015]. While existing approaches (PIC, FLIP, MPM) have proven very powerful, they suffer from some well-known limitations. The original PIC is stable, but highly dissipative, while FLIP, which is designed to eliminate this dissipation, is more noisy and at times unstable. We present a novel technique designed to retain the stability of the original PIC without suffering from the noise and instability

of FLIP. Our primary insight is that the dissipation in the original PIC results from a loss of information when transferring between the grid and particle representations. We prevent this loss of information by augmenting each particle with a locally affine (rather than a locally constant) description of the velocity. We show that this not only stably removes the dissipation of PIC, but that it also enables exact conservation of angular momentum across the transfers between the particles and the grid. With our method, we control noise by keeping the pure filter property of PIC but minimize information loss by enriching each particle with a tiny matrix providing a locally affine description of the flow. Our APIC method effectively reduces dissipation, preserves angular momentum and prevents instabilities. Furthermore, we demonstrate that our method is applicable to both incompressible liquids and MPM simulations.

Chapter 6 introduces a novel MPM for heat transport, melting, and solidifying materials [Stomakhin et al., 2014]. This brings a wider range of material behaviors into reach of the already versatile MPM. This is in contrast to best-of-breed fluid, deformable solid, or rigid-body solvers that are difficult to adapt to a wide range of materials. Extending the MPM requires several contributions. We introduce a dilational/deviatoric splitting of the constitutive model and show that an implicit treatment of the Eulerian evolution of the dilational part can be used to simulate arbitrarily incompressible materials. Furthermore, we show that this treatment reduces to a parabolic equation for moderate compressibility and an elliptic, Chorin-style projection at the incompressible limit. Since projections are naturally done on Marker-And-Cell (MAC) grids, we devise a staggered-grid MPM. Lastly, we adapt a heat-equation solver to the material point framework. The heat solver captures the underlying thermodynamics and alters material parameters. The method is implicit and capable of simulating nearly incompressible materials using a Chorin-like projection solve. Hence, we widen the range of materials that the MPM can handle, and demonstrate this range with several compelling melting and solidifying examples.

Appendix A shows that recasting the backward Euler method as a minimization problem allows Newton's method to be stabilized by standard optimization techniques with some novel improvements of our own. The resulting solver is capable of solving even the toughest simulations at the 24 Hz frame rate and beyond [Gast et al., 2015]. We show how simple collisions can be incorporated directly into the solver through constrained minimization without sacrificing efficiency. We also show that these techniques improve the behavior of MPM simulations by recasting the MPM as an optimization problem.

CHAPTER 2

The Material Point Method

The Material Point Method (MPM) requires both a Lagrangian and an Eulerian view of material deformation. In this chapter we briefly discuss the deformation theory and the basic algorithm for the MPM. We refer to [Bonet and Wood, 1997] for a more detailed introduction of continuum mechanics, and [Sulsky et al., 1995] for a more rigorous derivation of the MPM. In the context of Computer Graphics, [Stomakhin et al., 2013] is also a more appropriate introductory document for the MPM.

2.1 Deformation

We consider the motion of material to be determined by a mapping from material points $\mathbf{X}$ to a deformed configuration $\mathbf{x}$. One can define the deformation gradient as

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

2.1.1 Useful differentials

Differentials in terms of the deformation gradient $\boldsymbol{F}$ will appear often in the derivation of constitutive models and their derivatives (such as in Chapter 3). Here we list some common ones.

2.1.1.1 Differentials of the determinant

The determinant of $\boldsymbol{F}$ is $J(\boldsymbol{F})$. In both 2D and 3D, it can be explicitly written in terms of the entries of $\boldsymbol{F}$. A result worth remembering is

$$\delta J = J\boldsymbol{F}^{-T}\delta\boldsymbol{F},$$

or, directly, the derivative

$$\frac{\partial J}{\partial \boldsymbol{F}} = J\boldsymbol{F}^{-T}.$$

Furthermore, sometimes we need to compute $\delta(J\boldsymbol{F}^{-T}) = \frac{\partial(J\boldsymbol{F}^{-T})}{\partial \boldsymbol{F}} : \delta\boldsymbol{F}$. This is done by explicitly writing out each entry of the matrix $J\boldsymbol{F}^{-T}$, which is a polynomial of the entries of $\boldsymbol{F}$, and computing the differential directly.

2.1.1.2 Differentials of SVD

The SVD of $\boldsymbol{F}$ can be written as

$$\boldsymbol{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T,$$

where $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$.

Taking differentials yields

$$\delta\boldsymbol{F} = \delta\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T + \mathbf{U}\delta\boldsymbol{\Sigma}\mathbf{V}^T + \mathbf{U}\boldsymbol{\Sigma}\delta\mathbf{V}^T \tag{2.1}$$

$$\mathbf{0} = \delta\mathbf{U}^T\mathbf{U} + \mathbf{U}^T\delta\mathbf{U} \tag{2.2}$$

$$\mathbf{0} = \delta\mathbf{V}^T\mathbf{V} + \mathbf{V}^T\delta\mathbf{V}. \tag{2.3}$$

From (2.1) we have

$$\begin{aligned}\delta\mathbf{\Sigma} &= \mathbf{U}^T\delta\boldsymbol{F}\mathbf{V} - \mathbf{U}^T\delta\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V} - \mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\delta\mathbf{V}^T\mathbf{V} \\ &= \mathbf{U}^T\delta\boldsymbol{F}\mathbf{V} - \mathbf{U}^T\delta\mathbf{U}\mathbf{\Sigma} - \mathbf{\Sigma}\delta\mathbf{V}^T\mathbf{V}. \end{aligned} \tag{2.4}$$

From (2.2) and (2.3) we know that $\mathbf{U}^T\delta\mathbf{U}$ and $\delta\mathbf{V}^T\mathbf{V}$ are skew-symmetric, so their diagonal entries are 0. Therefore,

$$\begin{aligned}\delta\mathbf{\Sigma} &= \mathbf{U}^T\delta\boldsymbol{F}\mathbf{V} - \mathbf{U}^T\delta\mathbf{U}\mathbf{\Sigma} - \mathbf{\Sigma}\delta\mathbf{V}^T\mathbf{V} \\ &= \operatorname{diag}\left(\mathbf{U}^T\delta\boldsymbol{F}\mathbf{V} - \mathbf{U}^T\delta\mathbf{U}\mathbf{\Sigma} - \mathbf{\Sigma}\delta\mathbf{V}^T\mathbf{V}\right) \\ &= \operatorname{diag}\left(\mathbf{U}^T\delta\boldsymbol{F}\mathbf{V}\right).\end{aligned}$$

Substituting $\delta\mathbf{\Sigma}$ back into (2.4), if we write

$$\mathbf{U}^T\delta\mathbf{U} = \begin{pmatrix} 0 & x & y \\ -x & 0 & z \\ -y & -z & 0 \end{pmatrix}$$

and

$$\mathbf{V}^T\delta\mathbf{V} = \begin{pmatrix} 0 & r & s \\ -r & 0 & t \\ -s & -t & 0 \end{pmatrix},$$

then (2.4) is a linear system with 6 equations in 6 unknowns: x, y, z, r, s, and t. After solving for these unknowns, we can obtain $\delta\mathbf{U}$ and $\delta\mathbf{V}$.

2.1.1.3 Differentials of the Polar Decomposition

The polar decomposition of $\boldsymbol{F}$ can be written as

$$\boldsymbol{F} = \boldsymbol{R}\mathbf{S},$$

where $\mathbf{S} = \mathbf{S}^T$ and $\boldsymbol{R}\boldsymbol{R}^T = \mathbf{I}$. Similarly to $\mathbf{U}$ in the SVD case, $\delta\boldsymbol{R}^T\boldsymbol{R}$ is skew-symmetric with diagonal entries of 0.

Starting from the definitions, we have

$$\delta\boldsymbol{F} = \boldsymbol{R}\delta\mathbf{S} + \delta\boldsymbol{R}\mathbf{S} \tag{2.5}$$

$$\mathbf{0} = \delta\boldsymbol{R}\boldsymbol{R}^T + \boldsymbol{R}\delta\boldsymbol{R}^T \tag{2.6}$$

$$\delta\mathbf{S} = \delta\mathbf{S}^T. \tag{2.7}$$

Multiplying both sides of (2.5) with $\boldsymbol{R}^T$ gives

$$\boldsymbol{R}^T\delta\boldsymbol{F} = \delta\mathbf{S} + \boldsymbol{R}^T\delta\boldsymbol{R}\mathbf{S}. \tag{2.8}$$

Its transpose is

$$\delta\boldsymbol{F}^T\boldsymbol{R} = \delta\mathbf{S}^T + \mathbf{S}^T\delta\boldsymbol{R}^T\boldsymbol{R}. \tag{2.9}$$

Subtracting (2.8) and (2.9) yields

$$\begin{aligned}\boldsymbol{R}^T\delta\boldsymbol{F} - \delta\boldsymbol{F}^T\boldsymbol{R} &= \delta\mathbf{S} + \boldsymbol{R}^T\delta\boldsymbol{R}\mathbf{S} - \delta\mathbf{S}^T - \mathbf{S}^T\delta\boldsymbol{R}^T\boldsymbol{R} \\ &= \boldsymbol{R}^T\delta\boldsymbol{R}\mathbf{S} + \mathbf{S}\boldsymbol{R}^T\delta\boldsymbol{R}.\end{aligned}$$

This is a solvable linear system in x, y, and z if we assume

$$\boldsymbol{R}^T \delta \boldsymbol{R} = \begin{pmatrix} 0 & x & y \\ -x & 0 & z \\ -y & -z & 0 \end{pmatrix}.$$

After solving for x, y, and z, we can obtain $\delta \boldsymbol{R} = \boldsymbol{R}(\boldsymbol{R}^T \delta \boldsymbol{R})$ and substitute the result back into (2.8) to obtain $\delta \mathbf{S}$.

2.2 The MPM algorithm

In this section, we specify the full basic MPM algorithm in one time step. This algorithm assumes the most basic implementation of MPM. Both explicit time integration (easy to implement but requires smaller time step sizes) and the semi-implicit update scheme (requires solving a linear system but allows for larger time steps) as in [Stomakhin et al., 2013] will be given. All data structures are pre-allocated to proper sizes in memory. For notational simplicity, the algorithm assumes a fixed corotated constitutive model with no plasticity.

1. Grid data are reinitialized to default values of 0. This includes nodal mass m_i, nodal velocity $\boldsymbol{v}_i$, and all other helper data structures for solving the integration scheme.

2. For all particles, the interpolation weights w_{ip} and weight gradients ∇w_{ip} are computed and stored on the particles.

3. Mass and momentum are transferred from the particles to the grid according to

$$m_i^n = \sum_p w_{ip}^n m_p, \quad m_i^n \boldsymbol{v}_i^n = \sum_p w_{ip}^n m_p \boldsymbol{v}_p^n,$$

and nodal velocities are then computed as

$$\boldsymbol{v}_i^n = \frac{m_i^n \boldsymbol{v}_i^n}{m_i^n}.$$

Note that zero-mass nodes must be taken care of to prevent divide-by-zero problems.

4. The explicit forces on nodes are computed as

$$-\mathbf{f}_i^n(\boldsymbol{x}_i) = \sum_p V_p^0 \left(\frac{\partial \Psi}{\partial \boldsymbol{F}}(\boldsymbol{F}_p^n) \right) (\boldsymbol{F}_p^n)^T \nabla w_{ip}^n.$$

5. The explicit nodal velocity update is performed with

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i^n + \Delta t m_i^{-1} \mathbf{f}_i^n.$$

6. A grid-based collision (against solid boundary walls) is performed on $\boldsymbol{v}_i^*$. Depending on the type of collision, there can be different treatments of the velocity. For example, in the case of sticky material, one can simply set $\boldsymbol{v}_i^*$ to zero when the node is in contact or inside a collision object. In this step, one can also apply friction effects involving the normal and tangential components of $\boldsymbol{v}_i^*$.

7. For explicit time integration, one can set $\boldsymbol{v}_i^{n+1} = \boldsymbol{v}_i^*$ and directly jump to the next step. For implicit backward Euler time integration, a mass symmetric linear system

$$\sum_j \left(\mathbf{I}\delta_{ij} + \Delta t^2 m_i^{-1} \frac{\partial^2 \Phi^n}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_j} \right) \boldsymbol{v}_j^{n+1} = \boldsymbol{v}_i^*$$

is formed and Conjugate Residual (CR) or MINRES can be used to solve it.

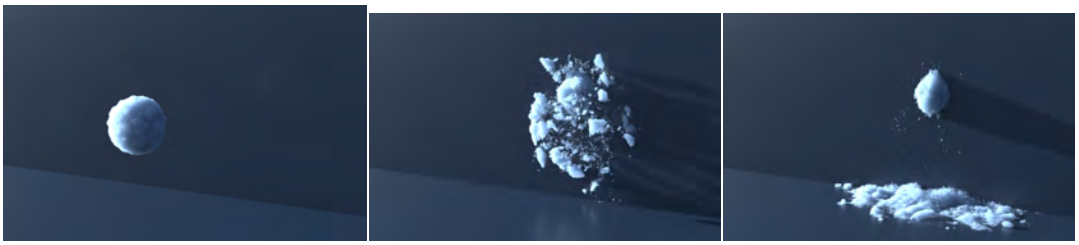

Figure 2.1: *Our approach works naturally with the MPM simulations from [Stomakhin et al., 2013]. Here we demonstrate this with a a snowball that smashes into a wall and sticks to it. Notably, we provide a new treatment of particle position updates that naturally prevents penetration in solid objects like the wall.*

8. Particle states are updated from grid velocities. In particular,

$$\boldsymbol{F}_p^{n+1} = (\mathbf{I} + \Delta t \sum_i \boldsymbol{v}_i^{n+1} (\nabla w_{ip}^n)^T) \boldsymbol{F}_p^n$$
$$\boldsymbol{v}_p^{n+1} = (1-\alpha) \sum_i \boldsymbol{v}_i^{n+1} w_{ip}^n + \alpha (\boldsymbol{v}_p^n + \sum_i (\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n) w_{ip}^n)$$
$$\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \sum_i \boldsymbol{v}_i^{n+1} w_{ip}^n.$$

   Here the velocity update is done with a combination of PIC and FLIP. We show in Chapter 5 that an easy modification can be done to it that largely improves accuracy and stability.

An improved transfer scheme will be given in Chapter 5. We develop an improved, optimization-based time integration scheme in Appendix A.

2.3 Snow simulation results

We demonstrate the advantages of using our improved integrator by applying it to the MPM snow formulation from [Stomakhin et al., 2013]. We ran three examples using both the original formulation and our modified formulation and compared with the snowball examples from the original paper. In each case, for our formulation we used

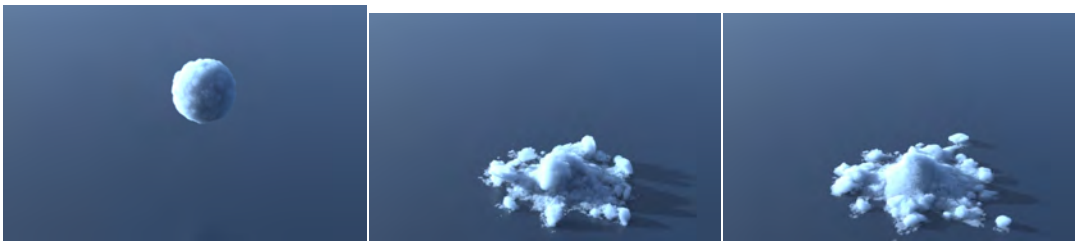

Figure 2.2: *Here we demonstrate with a snowball that drops to the ground and fractures.*

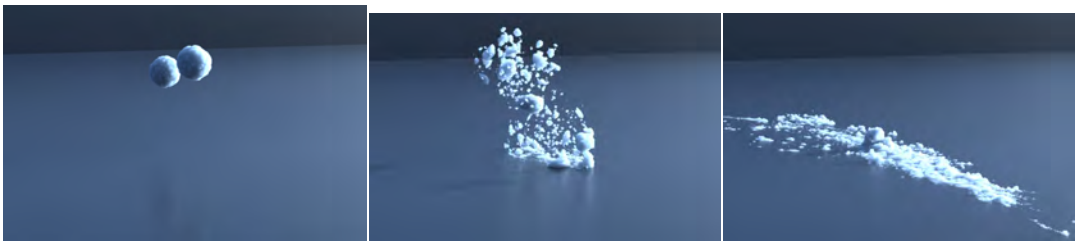

Figure 2.3: *The extension of our method to [Stomakhin et al., 2013] is robust to large deformation and collisions scenarios. Here we demonstrate this for with two snowballs that collide and fall to the ground.*

the CFL $\nu = 0.6$. Figure 2.1 shows a snowball hitting a wall using sticky collisions, which causes the snow to stick to the wall. Figure 2.2 shows a dropped snowball hitting the ground with sticky collisions. Figure 2.3 shows two snowballs colliding in mid air with sticky collisions against the ground. On average, we observed a speed up of 3.5 times over the original method. These results are tabulated in Table 2.1. Notably, we are able to take significantly larger time steps, however some of the potential gains from this are compromised by an increased complexity per time step. Nonetheless, our improved, optimization-based integrator provides a significant computational savings with minimal modification to the original approach.

Figure	Min Δt (s)		Average Δt (s)		Time/frame (s)		Speedup	Grid	# of
	Ours	Orig	Ours	Orig	Ours	Orig	factor	size	particles
2.2	6.0×10^{-4}	5.2×10^{-5}	3.4×10^{-3}	4.4×10^{-4}	59	184	3.1	$600 \times 300 \times 600$	2.8×10^5
2.1	5.4×10^{-4}	1.7×10^{-6}	1.5×10^{-3}	1.4×10^{-4}	85	431	5.1	$200 \times 240 \times 600$	2.8×10^5
2.3	3.5×10^{-4}	3.7×10^{-5}	1.6×10^{-3}	1.7×10^{-4}	288	780	2.7	$800 \times 300 \times 800$	5.6×10^5

Table 2.1: *Performance comparison of our modified MPM snow formulation ("Ours") with the original formulation ("Orig").*

CHAPTER 3

Elasticity and Plasticity

Terzopoulos et al. pioneered the use of elasticity, viscoelasticity, plasticity, and fracture in computer graphics [Terzopoulos et al., 1987; Terzopoulos and Fleischer, 1988a,b].

In physics or engineering, the constitutive model of a material usually describes how force is related to deformation or how stress is related to strain in the material. Together with the governing physical equations (such as Newton's second law), it can be used to compute the material motion under applied external forces or boundary conditions. This chapter assumes basic knowledge about the definitions of the deformation gradient, strain, and stress.

3.1 Hyperelasticity

A hyperelastic material refers to an ideally elastic material whose constitutive relation can be derived from an energy density function. Given an potential energy density Ψ and a discretization of a continuum, one can usually write out the total potential energy Φ of a material and use it in deriving forces.

3.1.1 Mass-spring system

The mass-spring system is very common in Computer Graphics due to its simplicity. It is most popular for simulating cloth. Continuum material is modeled as a collection of M point masses with positions $\boldsymbol{x}_i$, for $i = 1, \ldots, M$, interconnected by a set of N linear springs with stiffnesses c_k and natural lengths l_k, for $k = 1, \ldots, N$. Denoting the vector of all mass positions as $\boldsymbol{x}$, the total potential energy of the system at any time is

$$\Phi(\boldsymbol{x}) = \sum_k \frac{1}{2} c_k (|\boldsymbol{x}_i - \boldsymbol{x}_j| - l_k)^2,$$

where $|\boldsymbol{x}_i - \boldsymbol{x}_j|$ is the length of deformed spring k that interconnects point mass i situated at $\boldsymbol{x}_i$ and point mass j situated at $\boldsymbol{x}_j$. Given the potential energy, internal forces and force derivatives can be derived as $\mathbf{f}_i = -\frac{\partial \Phi}{\partial \boldsymbol{x}_i}$ and $\frac{\partial \mathbf{f}_i}{\partial \boldsymbol{x}_j} = -\frac{\partial^2 \Phi}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_i}$. They are used in solving the governing equations.

3.1.2 Neo-Hookean model

Continuum mechanics prefers to define energy density functions from deformation fields. The neo-Hookean model has been widely used to describe isotropic elastic materials. It is the simplest model for describing large deformation. The energy density for neo-Hookean materials is given by

$$\Psi(\boldsymbol{F}) = \frac{\mu}{2}(F_{ij}F_{ji} - d) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2,$$

where d is the dimension (2 or 3), $\boldsymbol{F}$ is the deformation gradient, and $J = |\boldsymbol{F}|$. The constants μ and λ are typically set from the Young's modulus and Poisson's ratio of the material. The first Piola-Kirchoff stress $\boldsymbol{P}$ is related to the energy density Ψ as

$\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}}$. For neo-Hookean materials, it can be shown that

$$\boldsymbol{P} = \mu \boldsymbol{F} + (\lambda \ln J - \mu) \boldsymbol{F}^{-T}.$$

3.1.3 Fixed corotated model

The $\ln(J)$ term in the neo-Hookean material is problematic when J approaches zero or becomes negative. This can never happen in the real world; however, numerical simulations generally do not prevent such nonphysical deformations. For robustness, [Stomakhin et al., 2012] has developed a fixed Corotated model that remains valid under inverted configurations. The fixed Corotated energy density is written as

$$\Psi = \mu \|\boldsymbol{F} - \boldsymbol{R}\|_F^2 + \frac{\lambda}{2}(J-1)^2,$$

where $\boldsymbol{R}$ comes from the polar decomposition $\boldsymbol{F} = \boldsymbol{R}\mathbf{S}$. It can also be written in terms of the singular values of $\boldsymbol{F}$ as

$$\begin{aligned}\Psi(\boldsymbol{F}) = \hat{\Psi}(\boldsymbol{\Sigma}(\boldsymbol{F})) &= \mu \sum_i (\sigma_i - 1)^2 + \frac{\lambda}{2}(J-1)^2 \\ &= \mu \left(tr(\boldsymbol{F}^T \boldsymbol{F}) - 2 \sum_i \sigma_i + d \right) + \frac{\lambda}{2}(J-1)^2,\end{aligned}$$

where $\sigma_i = \boldsymbol{\Sigma}_{ii}$ comes from the polar singular value decomposition $\boldsymbol{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. Here the polar SVD is computed such that $\mathbf{U}$ and $\mathbf{V}$ are rotations. Also, $\boldsymbol{R} = \mathbf{U}\mathbf{V}^T$ and $\mathbf{S} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{V}^T$. Using $\delta\boldsymbol{\Sigma} = \mathrm{diag}(\mathbf{U}^T \delta \boldsymbol{F} \mathbf{V})$, we can show that the first Piola-Kirchoff stress is

$$\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}} = 2\mu(\boldsymbol{F} - \boldsymbol{R}) + \lambda(J-1)J\boldsymbol{F}^{-T}.$$

The second derivatives can be computed as

$$\frac{\partial^2 \Psi}{\partial \boldsymbol{F} \partial \boldsymbol{F}} : \delta \boldsymbol{F} = \delta \left(\frac{\partial \Psi}{\partial \boldsymbol{F}} \right) = 2\mu \delta \boldsymbol{F} - 2\mu \delta \boldsymbol{R} + \lambda J \boldsymbol{F}^{-T} \delta J + \lambda (J-1) \delta (J \boldsymbol{F}^{-T}).$$

Here $\delta J = J \boldsymbol{F}^{-T} \delta \boldsymbol{F}$ and $\delta (J \boldsymbol{F}^{-T}) = \frac{\partial (J \boldsymbol{F}^{-T})}{\partial \boldsymbol{F}} : \delta \boldsymbol{F}$. Also, $\delta \boldsymbol{R} = \boldsymbol{R} \boldsymbol{R}^T \delta \boldsymbol{R}$ can be computed using $\boldsymbol{F} = \boldsymbol{R}\mathbf{S}$ and $\mathbf{S} = \mathbf{S}^T$. This involves solving a linear system for $\boldsymbol{R}^T \delta \boldsymbol{R}$ which can be proven to be skew symmetric from $\boldsymbol{R}\boldsymbol{R}^T = \mathbf{I}$. For a full derivation, see the technical document of Stomakhin et al. [2013].

All our purely elastic materials, except for those in Chapter 6, are simulated with this model due to its simplicity and robustness.

3.2 Plasticity

This section presents the simple plasticity model proposed in [Stomakhin et al., 2013] for simulating the dynamic behaviors of snow. In the real world, snow behavior is very complicated due to its water-ice mixture nature. Rather than being accurate in its physical parameters, the model discussed in this section is designed to be simple yet achieve visual realism.

3.2.1 Plastic flow

The plastic flow is modeled as a constraint on the deformation gradient $\boldsymbol{F}$. In a general MPM framework, the treatment discussed below is performed at the end of each time step independently on the $\boldsymbol{F}$ stored on each particle. It also generally works for a finite element discretization as long as the deformation is represented with deformation gradients.

The plastic flow is achieved by decomposing $\boldsymbol{F}$ into an elastic part and a plastic part:

$\boldsymbol{F} = \boldsymbol{F}_E\boldsymbol{F}_P$, where the energy density function is defined only in terms of $\boldsymbol{F}_E$. With this definition, $\boldsymbol{F}_p$ represents the new local rest state of the material. This is similar to enforcing permanent deformation via changing the rest configuration.

In any time step $n+1$, we denote the previously stored deformation gradients on particles as $\boldsymbol{F}_E^n$ and $\boldsymbol{F}_P^n$. Here we emitted the particle index because the process is independently carried out per particle. By definition, $\boldsymbol{F}^n = \boldsymbol{F}_E^n\boldsymbol{F}_P^n$. We first assume the new deformation introduced in time step $n+1$ is purely elastic; that is, the MPM algorithm computes the deformation gradient update as

$$\hat{\boldsymbol{F}}_E^{n+1} = (\mathbf{I} + \Delta t \nabla \boldsymbol{v}^{n+1})\boldsymbol{F}_E^n.$$

Here, $\hat{\boldsymbol{F}}_E^{n+1}$ is temporary. We want to take part of it and make that plastic. First, we compute the total deformation gradient

$$\boldsymbol{F}^{n+1} = \hat{\boldsymbol{F}}_E^{n+1}\boldsymbol{F}_P^n.$$

Then, we compute the SVD of $\hat{\boldsymbol{F}}_E^{n+1}$ as

$$\hat{\boldsymbol{F}}_E^{n+1} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}.$$

We further clamp each entry of $\hat{\mathbf{\Sigma}}$ to the range $[l, h]$, where $l < 1 < h$. The closer l and h are to 1, the easier the material becomes plastic. In particular, h controls stretching and l controls compression. Denoting the clamped version of $\hat{\mathbf{\Sigma}}$ as $\mathbf{\Sigma}$, the final deformation gradient components are

$$\begin{aligned}\boldsymbol{F}_E^{n+1} &= \hat{\mathbf{U}}\mathbf{\Sigma}\hat{\mathbf{V}},\\ \boldsymbol{F}_P^{n+1} &= (\boldsymbol{F}_E^{n+1})^{-1}\boldsymbol{F}^{n+1}.\end{aligned}$$

3.2.2 Material hardening

In a continuum view, snow becomes harder during compression. In extension, it fractures much more easily than other plastic materials such as gum. To achieve these two effects, we can let the constitutive model parameters depend on the plastic deformation. In particular, for the fixed Corotated model introduced in Section 3.1.3, we quantify this hardening/softening effect as

$$\mu = \mu_0 e^{\xi(1-J_p)},$$
$$\lambda = \lambda_0 e^{\xi(1-J_p)},$$

where ξ is called the hardening factor and $J_p = |\boldsymbol{F}_P|$ measures the severity of the plastic deformation.

A starting parameter set for snow (as given in [Stomakhin et al., 2013]) is density $\rho = 400$, Young's modulus $E = 1.4 \times 10^5$, Poisson's ratio $\nu = 0.2$, lower plasticity clamping $l = 1 - 2.5 \times 10^{-2}$, higher plasticity clamping $h = 1 + 7.5 \times 10^{-3}$, and hardening factor $\xi = 10$. For dry sand, since it can freely separate, we set $h = 1$.

Section 5.5 presents some results in simulating dry sand. The snow simulation results are mainly demonstrated in [Stomakhin et al., 2013] and Disney's 2013 feature animation "*Frozen*".

3.3 Lagrangian forces

This section presents a method for discretizing any mesh-based Lagrangian force model to a standard MPM grid solver.

For most of the MPM simulations, we compute forces as in [Stomakhin et al., 2013]. This approach has the advantages of a mesh free method, such as effortless topology

change. For objects not intended to undergo topological changes, a meshed approach is also available using the ideas in [Sifakis et al., 2007]. In this case, we can use any Lagrangian force model (springs, finite elements, etc.) for which we can write down the total potential energy $\Phi(\boldsymbol{x}_p)$. Corresponding to this Lagrangian force model, we compute forces $\mathbf{f}_p = -\frac{\partial \Psi}{\partial \boldsymbol{x}_p}$. We also assume that, given a vector $\delta\mathbf{u}_q$ on particles, we can multiply by force derivatives $\frac{\partial \mathbf{f}_p}{\partial \boldsymbol{x}_q}$ to obtain $\delta\mathbf{f}_p = \sum_q \frac{\partial \mathbf{f}_p}{\partial \boldsymbol{x}_q}\delta\mathbf{u}_q$. Since these force-related constructs are purely Lagrangian and are computed in the usual way, we will not elaborate on them here.

Although we have defined our mesh-based forces as Lagrangian forces, we must still apply them through the grid. We must describe how particle positions $\boldsymbol{x}_p$ relate to our (conceptually) moving grid nodes $\boldsymbol{x}_i$ so that forces can be evaluated. Then, we must compute $\mathbf{f}_i$ from $\mathbf{f}_p$ and find a means of computing $\delta\mathbf{f}_i$ given $\delta\mathbf{u}_i$. Doing this allows us to use Lagrangian forces as Eulerian forces. Comparing our update rules for $\boldsymbol{x}_p$ and $\boldsymbol{x}_i$, we find $\boldsymbol{x}_p = \sum_i w_{ip}\boldsymbol{x}_i$. Using the chain rule,

$$\mathbf{f}_i = \sum_p w_{ip}\mathbf{f}_p \qquad \delta\mathbf{f}_i = \sum_{p,q,j} w_{ip}\frac{\partial \mathbf{f}_p}{\partial \boldsymbol{x}_q}w_{jq}\delta\mathbf{u}_j. \tag{3.1}$$

Since these forces are applied to the grid, both the MPM and Lagrangian approaches can be employed in the same simulation. Each particle is labeled as an MPM particle or a meshed particle. Note that the deformation gradient $\boldsymbol{F}_p$ stored on meshed particles is never used, since for those particles this quantity is computed using the mesh. This provides an effective means of coupling MPM with mesh-based approaches, such as shown in Figure 3.1. This gives the precise surface tracking of Lagrangian techniques coupled with the automatic collision handling of Eulerian grids.

A coupling example is shown in Figure 3.1. Here we use a traditional MPM discretization of the elastoplastic constitutive model from [Bargteil et al., 2007] to simulate frozen yogurt. We couple this with an elastic cloth using the Lagrangian force

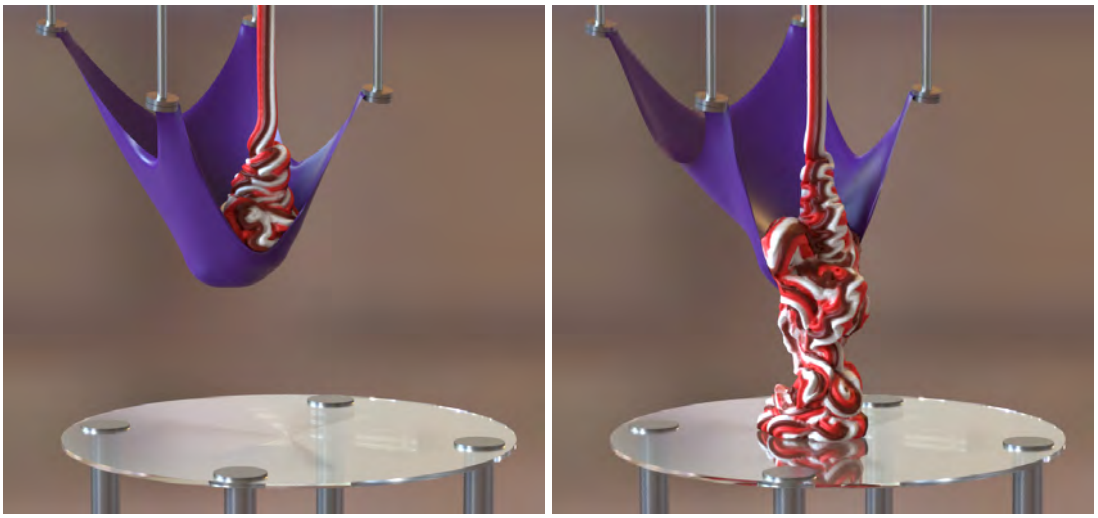

Figure 3.1: *An APIC coupled simulation of elastoplastic frozen yogurt and elastic cloth, where coupling is achieved using our MPM approach with Lagrangian energy-based forces.*

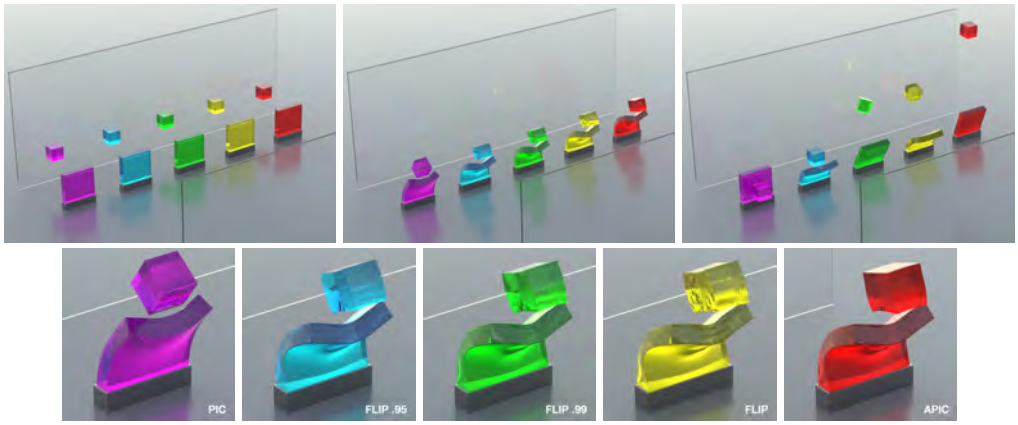

Figure 3.2: *We compare APIC with FLIP and PIC using the Lagrangian force model from Section 3.3 and a collision scenario with significant angular momentum. APIC preserves angular momentum better than even pure FLIP, and is at the same time the most stable of the various options. In the bottom row we show that the cube surface remains smooth after collision with APIC relative to the behavior of FLIP.*

model. The cloth is modeled using a standard mass-spring energy. In Figure 3.2 we show a comparison using mesh-based cubes with a Lagrangian finite element constitutive model. Here, APIC retains angular momentum and energetic behavior better than PIC, FLIP, and FLIP/PIC blends. FLIP and FLIP/PIC blends produce ringing during the collisions between the cube and the glass plates and flexible block.

CHAPTER 4

Simulating Viscoelastic Fluids, Foams and Sponges

In this chapter, we present a constitutive model for simulating viscoelastic fluids, foams and sponges. We design our discretization from the upper convected derivative terms in the evolution of the left Cauchy-Green elastic strain tensor. We combine this with an Oldroyd-B model for plastic flow in a complex viscoelastic fluid. While the Oldroyd-B model is traditionally used for viscoelastic fluids, we show that its interpretation as a plastic flow naturally allows us to simulate a wide range of complex material behaviors. In order to do this, we provide a modification to the traditional Oldroyd-B model that guarantees volume preserving plastic flows. Our plasticity model is remarkably simple (foregoing the need for the singular value decomposition (SVD) of stresses or strains).

4.1 Background

Non-Newtonian fluid behavior is exhibited by a wide range of everyday materials including paint, gels, sponges, foams and various food components like ketchup and custard [Larson, 1999]. These materials are often special kinds of colloidal systems (a type of mixture in which one substance is dispersed evenly throughout another), where dimensions exceed those usually associated with colloids (up to 1μm for the dispersed phase) [Hiemenz and Rajagopalan, 1997; Larson, 1999]. For example, when a gas and a liquid are shaken together, the gas phase becomes a collection of bubbles dispersed in the liquid: This is the most common observation of foams. While a standard Newtonian

viscous stress is a component of the mechanical response of these materials, they are non-Newtonian in the sense that there are other, often elastoplastic, aspects of the stress response to flow rate and deformation. Comprehensive reviews are given in [Morrison and Ross, 2002; Prudhomme and Kahn, 1996; Schramm, 1994; Larson, 1999].

Terzopoulos and Fleischer were the first in computer graphics to show the effects possible with simulated elastoplastic materials [Terzopoulos and Fleischer, 1988a,b]. Since those seminal works, many researchers have developed novel methods capable of replicating a wide range of material behaviors. Generally, these fall into one of three categories: Eulerian grid, Lagrangian mesh, or particle-based techniques.

Eulerian grid-based approaches: Goktekin et al. [2004] showed that the addition of an Eulerian elastic stress with von Mises criteria plasticity to the standard level-set-based simulation of free surface Navier Stokes flows can capture a wide range of viscoelastic behaviors. Losasso et al. [2006a] also use an Eulerian approach. Rasmussen et al. [2004] experiment with a range of viscous effects for level-set-based free surface melting flows. Batty and Bridson [2008a]; Batty and Houston [2011] use Eulerian approaches to efficiently simulate spatially varying viscous coiling and buckling. Carlson et al. [2002a] also achieve a range of viscous effects.

Lagrangian mesh-based approaches: Lagrangian methods naturally resolve deformation needed for elastoplasticity; however, large strains can lead to mesh tangling for practical flow scenarios and remeshing is required. Bargteil et al. [2007] show that this can achieve impressive results. This was later extended to embedded meshes in [Wojtan and Turk, 2008] and further treatment of splitting and merging was achieved in [Wojtan et al., 2009]. Batty et al. [2012] used a reduced dimension approach to simulate thin viscous sheets with adaptively remeshed triangle meshes.

Particle Methods: Ever since Desbrun and Gascuel [1996] showed that SPH can be used for a range of viscous behavior, particle methods have been popular for achieving

complex fluid effects. Like Gotekin et al., Chang et al. [2009] also use an Eulerian update of the strain for elastoplasic SPH simulations. Solenthaler et al. [2007] show that SPH can be used to compute strain and use this to get a range of elastoplastic effects. Becker et al. [2009] show that this can be generalized to large rotational motion. [Gerszewski et al., 2009] also update deformation directly on particles. Keiser et al. [2005] and Müller et al. [2004] also add elastic effects into SPH formulations. Paiva et al. [2006, 2009] use a non-Newtonian model for fluid viscosity.

Although the MPM is a hybrid grid/particle method, particles are arguably the primary material representation. The MPM has recently been used to simulate elastoplastic flows to capture snow in [Stomakhin et al., 2013] and varied, melting materials in [Stomakhin et al., 2014]. Also, Yue et al. [2015] use the MPM to simulate Herschel-Bulkley plastic flows for foam.

We show that the MPM approach can be generalized to achieve a wide range of viscoelastic, complex fluid effects. Most computer graphics approaches use a von Mises like stress or strain-based plastic flow criteria which require expensive SVD computations. With our Oldroyd-inspired approach, we avoid the need for the SVD of either elastic or plastic responses. We show that this comparatively simple model gives rise to a wide range of interesting non-Newtonian material behaviors.

4.2 Governing equations

The governing equations arise from basic conservation of mass and momentum as

$$\frac{D}{Dt}\rho + \rho\nabla\cdot\boldsymbol{v} = 0, \quad \rho\frac{D}{Dt}\boldsymbol{v} = \nabla\cdot\boldsymbol{\sigma} + \rho\mathbf{g}, \tag{4.1}$$

where ρ is the mass density, $\boldsymbol{v}$ is the velocity, $\boldsymbol{\sigma}$ is the Cauchy stress and $\mathbf{g}$ is the gravitational acceleration. As is commonly done with viscoelastic complex fluids, we

write the Cauchy stress as $\boldsymbol{\sigma} = \boldsymbol{\sigma}^N + \boldsymbol{\sigma}^E$, where $\boldsymbol{\sigma}^N = \frac{\mu^N}{2}\left(\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T\right)$ is the viscous Newtonian component and $\boldsymbol{\sigma}^E$ is the elastic component. We express the constitutive behavior through the elastic component of the left Cauchy Green strain. Specifically, the deformation gradient of the flow $\boldsymbol{F}$ can be decomposed as a product of elastic and plastic deformation as $\boldsymbol{F} = \boldsymbol{F}^E\boldsymbol{F}^P$ and the elastic left Cauchy Green strain is $\mathbf{b}^E = \boldsymbol{F}^E(\boldsymbol{F}^E)^T$ [Bonet and Wood, 1997]. With this convention, we can define the elastic portion of the Cauchy stress via the stored elastic potential $\psi(\mathbf{b}^E)$ as $\boldsymbol{\sigma}^E = \frac{2}{J}\frac{\partial \psi}{\partial \mathbf{b}^E}\mathbf{b}^E$.

4.2.1 Upper convected derivative

We can define the plastic flow using the temporal evolution of the elastic right Cauchy Green strain as in [Bonet and Wood, 1997]. Rewriting $\boldsymbol{F}^E = \boldsymbol{F}(\boldsymbol{F}^P)^{-1}$, we have $\mathbf{b}^E = \boldsymbol{F}(\mathbf{C}^P)^{-1}\mathbf{F}^T$, where $\mathbf{C}^P = (\boldsymbol{F}^P)^T\boldsymbol{F}^P$ is the right plastic Cauchy Green strain. The Eulerian form of the temporal evolution is then obtained by taking the material derivative

$$\frac{D\mathbf{b}^E}{Dt} = \frac{D\boldsymbol{F}}{Dt}(\mathbf{C}^P)^{-1}\boldsymbol{F}^T + \boldsymbol{F}(\mathbf{C}^P)^{-1}\frac{D\boldsymbol{F}}{Dt}^T + \boldsymbol{F}\frac{D}{Dt}[(\mathbf{C}^P)^{-1}]\boldsymbol{F}^T. \tag{4.2}$$

With this view, the plastic flow is defined via $\frac{D}{Dt}[(\mathbf{C}^P)^{-1}]$. This is relatively difficult when using von Mises style plastic yield criteria and it is more straightforward to work directly with $\boldsymbol{F}^E$ and $\boldsymbol{F}^P$ in that case. However, interpreting the Oldroyd model as a plastic flow can be seen as $\frac{D}{Dt}[(\mathbf{C}^P)^{-1}] = \frac{1}{Wi}(\mathbf{C}^{-1} - (\mathbf{C}^P)^{-1})$, where $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ is the right Cauchy Green strain. Combining this with $\frac{D}{Dt}\boldsymbol{F} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\boldsymbol{F}$ (e.g. [Bonet and Wood, 1997]), we obtain

$$\frac{D\mathbf{b}^E}{Dt} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}^E + \mathbf{b}^E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T + \frac{1}{Wi}(\mathbf{I} - \mathbf{b}^E). \tag{4.3}$$

We can see, both from the $\frac{1}{Wi}(\mathbf{I} - \mathbf{b}^E)$ and $\frac{D}{Dt}[(\mathbf{C}^P)^{-1}] = \frac{1}{Wi}(\mathbf{C}^{-1} - (\mathbf{C}^P)^{-1})$ terms that the plasticity achieves a strong damping of the elastic component of the stress. The

severity of this damping is inversely proportional to the Weissenberg number Wi. This equation is often abbreviated as

$$\overset{\triangledown}{\mathbf{b}}^E = \frac{1}{Wi}(\mathbf{I} - \mathbf{b}^E), \tag{4.4}$$

where the operator $\overset{\triangledown}{\mathbf{b}}^E$ is defined to be $\overset{\triangledown}{\mathbf{b}}^E \equiv \frac{D}{Dt}\mathbf{b}^E - \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{b}^E - \mathbf{b}^E\frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T$. This is often referred to as the upper convected derivative and it appears in many mathematical descriptions of complex fluids [Larson, 1999].

4.2.2 Volume-preserving plasticity

The plastic flow in the Oldroyd model will not generally be volume-preserving. Since many plastic flows, including those of foams, exhibit this behavior, we provide a modification to the standard Oldroyd model that will satisfy this. If we define $\mathbf{b}^E_{OB}$ to obey $\overset{\triangledown}{\mathbf{b}}^E_{OB} = \frac{1}{Wi}(\mathbf{I} - \mathbf{b}^E_{OB})$, then we define a new elastic left Cauchy Green stress as $\mathbf{b}^E \equiv \left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}}\mathbf{b}^E_{OB}$, where $J = \det(\boldsymbol{F})$ and $J_{OB} = \sqrt{\det(\mathbf{b}^E_{OB})}$. Using this definition, $\det(\mathbf{b}^E) = J^2$ and since by definition $\det(\mathbf{b}^E) = \det(\boldsymbol{F}^E)^2$ and $J = \det(\boldsymbol{F}^E)\det(\boldsymbol{F}^P)$, we see that it must be true that $\det(\boldsymbol{F}^P) = 1$, and thus the plastic flow is volume-preserving. This modification to the Oldroyd plasticity obeys

$$\overset{\triangledown}{\mathbf{b}}^E = \frac{D}{Dt}\left(\left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}}\right)\mathbf{b}^E_{OB} + \frac{1}{Wi}\left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}}(\mathbf{I} - \mathbf{b}^E_{OB}), \tag{4.5}$$

which has the plastic flow

$$\begin{aligned}\frac{D}{Dt}[(\mathbf{C}^P)^{-1}] = \frac{D}{Dt}\left(\left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}}\right)(\mathbf{C}^P_{OB})^{-1} + \\ \frac{1}{Wi}\left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}}(\mathbf{C}^{-1} - (\mathbf{C}^P_{OB})^{-1}).\end{aligned} \tag{4.6}$$

We need not solve for $\mathbf{b}^E$ using the definition of its plastic flow. In practice, we solve for the comparatively simple $\mathbf{b}^E_{OB}$ and then obtain the elastic stress as $\mathbf{b}^E = \left(\frac{J}{J_{OB}}\right)^{\frac{2}{3}} \mathbf{b}^E_{OB}$. We provide this derivation here only to show that there is a plastic flow associated with this definition of the elastic stress.

4.2.3 Elasticity

We define constitutive behavior through the compressible neo-Hookean elastic potential energy density as

$$\psi(\mathbf{b}^E) = \frac{\mu}{2}(\text{tr}(\mathbf{b}^E) - 3) - \mu \ln(J) + \frac{\lambda}{2}(J-1)^2 \tag{4.7}$$

with associated Cauchy stress

$$\boldsymbol{\sigma}^E = \frac{\mu}{J}(\mathbf{b}^E - \mathbf{I}) + \lambda(J-1)\mathbf{I}. \tag{4.8}$$

4.3 Discretization

We closely follow the algorithm from [Stomakhin et al., 2013] and Section 2.2. The only difference is in the discrete Eulerian grid node forces and force derivatives. All steps in the algorithm not related to the update of grid node velocities are the same; we simply change the nature of stress-based forces. In this section, we describe how to modify the potential-based definition of these forces to discretize our new governing equations.

As before, we denote the position, velocity and deformation gradient of particle p at time t^n as $\boldsymbol{x}_p^n$, $\boldsymbol{v}_p^n$, and $\boldsymbol{F}_p^n$, respectively. Eulerian grid node locations are denoted as $\boldsymbol{x}_{\mathbf{i}}$, where $\mathbf{i} = (i,j,k)$ is the grid node index. The weights at time t^n are $w_{\mathbf{i}p}^n = N_{\mathbf{i}}(\boldsymbol{x}_p^n)$, where $N_{\mathbf{i}}(\boldsymbol{x})$ is the interpolation function associated with grid node $\mathbf{i}$ and the weight

gradients are $\nabla w_{\mathbf{i}p}^n = \nabla N_{\mathbf{i}}(\boldsymbol{x}_p^n)$. We define the forces on the Eulerian grid nodes as the derivative of an energy with respect to grid node locations. We do not actually move grid nodes, but we consider their movement to define grid node velocities $\boldsymbol{v}_{\mathbf{i}}$ as $\hat{\boldsymbol{x}}_{\mathbf{i}} = \boldsymbol{x}_{\mathbf{i}} + \Delta t \boldsymbol{v}_{\mathbf{i}}$. Using $\hat{\boldsymbol{x}}$ to denote the vector of all grid nodes, we define the potential

$$\Phi(\hat{\boldsymbol{x}}) = \sum_p \left(\Phi^E(\hat{\boldsymbol{x}})V_p^0 + \Phi^N(\hat{\boldsymbol{x}})V_p^n\right), \tag{4.9}$$

where $\Phi^E(\hat{\boldsymbol{x}})$ is the elastoplastic component of the potential energy density $\Phi^E(\hat{\boldsymbol{x}}) = \psi\left(\hat{\mathbf{b}}^E(\hat{\boldsymbol{x}})\right)$ and $\Phi^N(\hat{\boldsymbol{x}})$ is the Newtonian viscous potential energy density

$$\Phi^N(\hat{\boldsymbol{x}}) = \mu^N \hat{\boldsymbol{\epsilon}}_p(\hat{\boldsymbol{x}}) : \hat{\boldsymbol{\epsilon}}_p(\hat{\boldsymbol{x}}) = \sum_{i,j} \mu^N \hat{\epsilon}_{p_{ij}}(\hat{\boldsymbol{x}})\hat{\epsilon}_{p_{ij}}(\hat{\boldsymbol{x}}). \tag{4.10}$$

Here $\hat{\boldsymbol{\epsilon}}_p(\hat{\boldsymbol{x}}) = \frac{1}{2}\left(\nabla\hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}) + (\nabla\hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}))^T\right)$ is the strain rate at $\boldsymbol{x}_p^n$ induced by the grid node motion defined by $\hat{\boldsymbol{x}}$ over the time step, $\nabla\hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}) = \sum_{\mathbf{i}} \frac{\hat{\boldsymbol{x}}_{\mathbf{i}} - \boldsymbol{x}_{\mathbf{i}}}{\Delta t}(\nabla w_{\mathbf{i}p}^n)^T$, and V_p^0 is the volume of the material originally occupied by the particle p. However, for the viscous Newtonian potential, we are approximating an integral over the time t^n configuration of the material, so we have $V_p^n = \det(\boldsymbol{F}_p^n)V_p^0$.

A deformation gradient $\boldsymbol{F}_p^n$ is stored on each particle and updated using

$$\hat{\boldsymbol{F}}(\hat{\boldsymbol{x}}) = (\mathbf{I} + \Delta t \nabla\hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}))\boldsymbol{F}_p^n. \tag{4.11}$$

We use this to define $\hat{J}_p(\hat{\boldsymbol{x}}) = \det(\hat{\boldsymbol{F}}(\hat{\boldsymbol{x}}))$ in the definition of

$$\hat{\mathbf{b}}^E(\hat{\boldsymbol{x}}) = \left(\frac{\hat{J}_p(\hat{\boldsymbol{x}})^2}{\det(\hat{\mathbf{b}}^E_{OB_p}(\hat{\boldsymbol{x}}))}\right)^{\frac{1}{3}} \hat{\mathbf{b}}^E_{OB_p}(\hat{\boldsymbol{x}}). \tag{4.12}$$

Similar to the treatment in (4.11), we store $\mathbf{b}^{E^n}_{OB_p}$ on each particle and discretize the

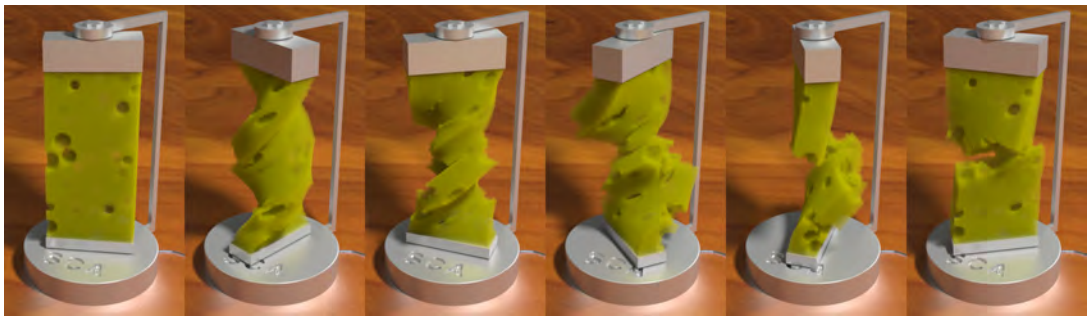

Figure 4.1: *A soft sponge is twisted. It fractures and collides with itself. The failure and contact phenomena are resolved automatically by the MPM approach.*

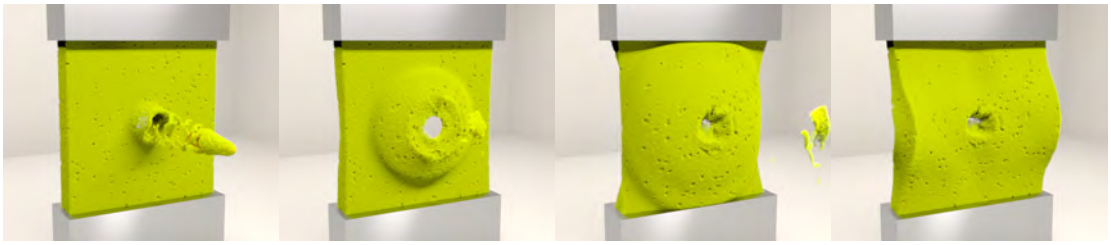

Figure 4.2: *A kinematic bullet is fired at a sponge, resulting in significant deformation and fracture.*

upper convected derivative terms in the evolution equation for $\mathbf{b}_{OB}^{E}$ to obtain

$$\begin{aligned}\hat{\mathbf{b}}_{OB_p}^{E}(\hat{\boldsymbol{x}}) =& \Delta t \nabla \hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}) \mathbf{b}_{OB_p}^{E^n} + \Delta t \mathbf{b}_{OB_p}^{E^n} (\nabla \hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}))^T \\ &+ \frac{\Delta t}{Wi}\mathbf{I} + \left(1 - \frac{\Delta t}{Wi}\right)\mathbf{b}_{OB_p}^{E^n}.\end{aligned} \tag{4.13}$$

The force on the grid nodes is defined as $\mathbf{f}(\hat{\boldsymbol{x}}) = -\frac{\partial \Phi}{\partial \hat{\boldsymbol{x}}}(\hat{\boldsymbol{x}})$ and it is used in the implicit update of grid velocities $\mathbf{v}_{\mathbf{i}}^{n+1}$, exactly as in Chapter 2. We work out these derivatives as well as the $\frac{\partial \mathbf{f}}{\partial \hat{\boldsymbol{x}}}(\hat{\boldsymbol{x}})$ in Appendix B.

4.4 Simulation results

In Figure 4.1, a sponge is twisted with top and bottom fixed by Dirichlet boundary conditions. Dynamic fracture and self collision are naturally handled. In Figure 4.2,

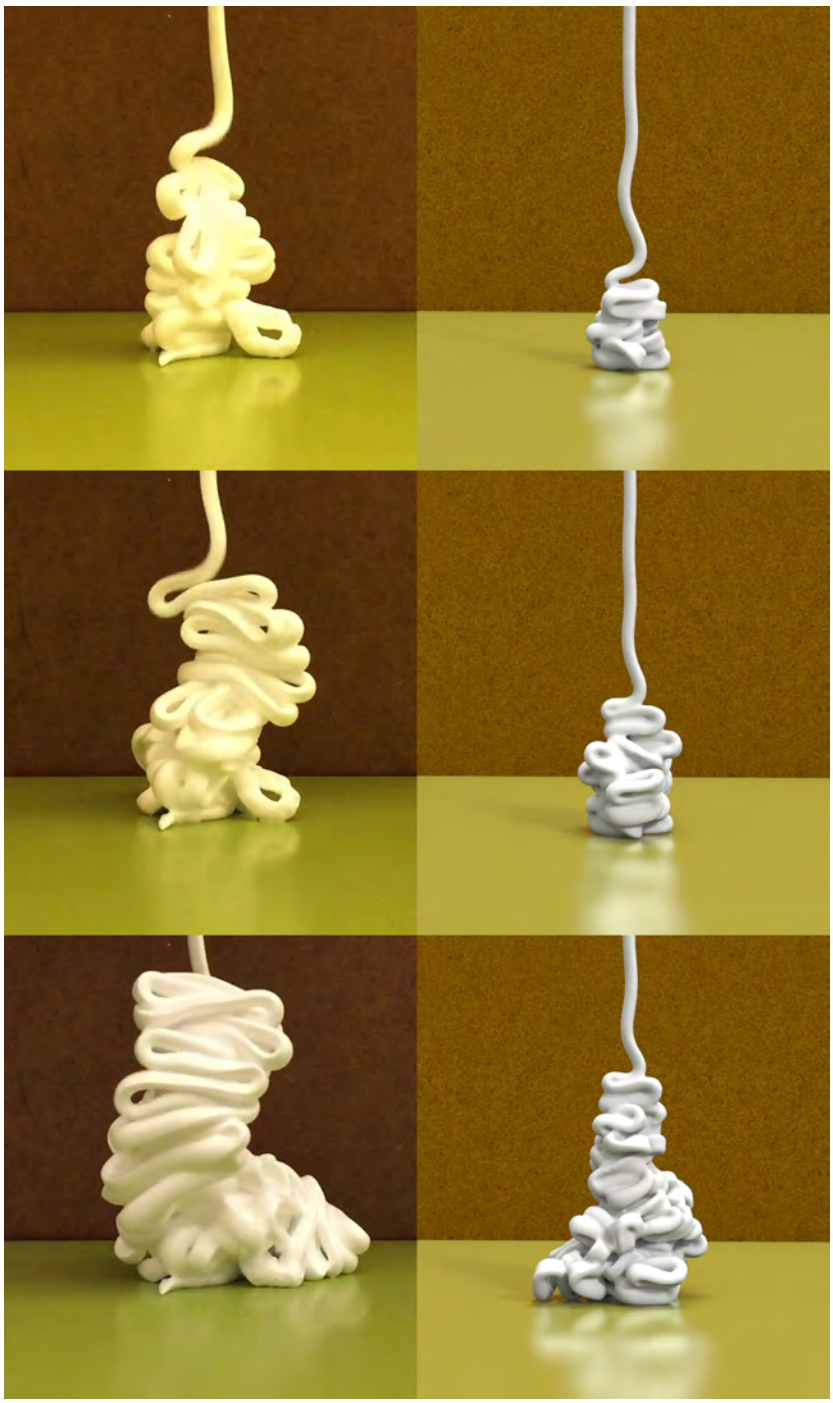

Figure 4.3: *Simulated shaving foam (right) is compared with real world footage (left). The simulation captures the characteristic S-shaped buckling and elastic behavior.*

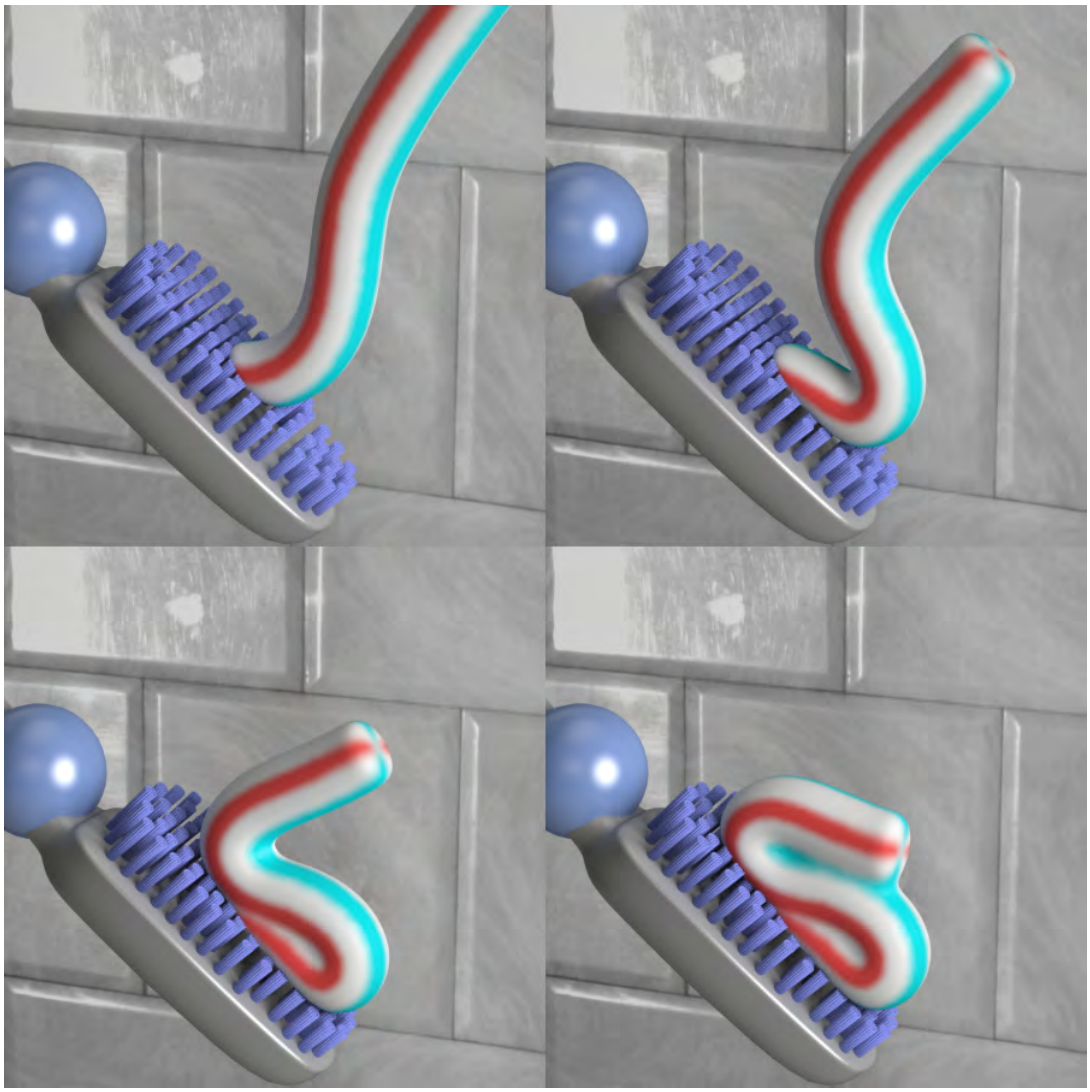

Figure 4.4: *A simulation of toothpaste. Unlike the shaving foam, Newtonian viscosity dominates material behavior.*

	ρ	μ	λ	μ^N	Wi
Twisting sponge	2	3.6×10^2	1.4×10^3	0	50
Shooting sponge	1	3.6×10^2	1.4×10^3	0	50
Shaving foam	0.2	5	50	1×10^{-4}	0.5
Toothpaste	1	0.839	8.39	1×10^{-1}	0.4
Viennetta ice cream	1	1	10	5×10^{-5}	0.1
Pie cream	0.2	5	50	1×10^{-7}	1×10^{-4}
Pie crust	0.5	5×10^5	4×10^6	1×10^{-8}	1×10^{30}
Pie crust scored	0.5	5	10	1×10^{-5}	1

Table 4.1: *Material parameters.*

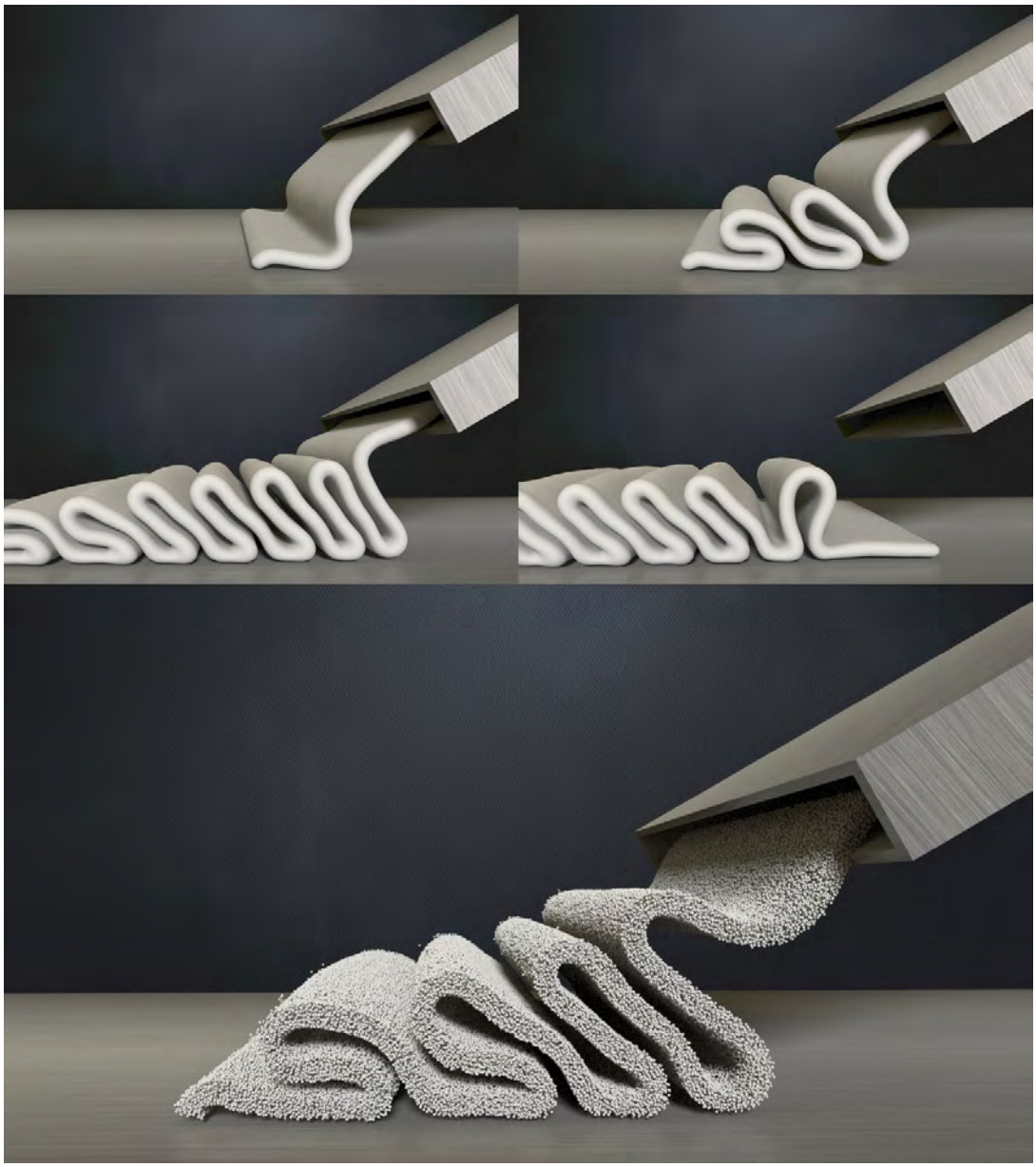

Figure 4.5: *Simulated Viennetta ice cream is poured onto a conveyor belt and forms characteristic folds. A particle view is shown on the bottom.*

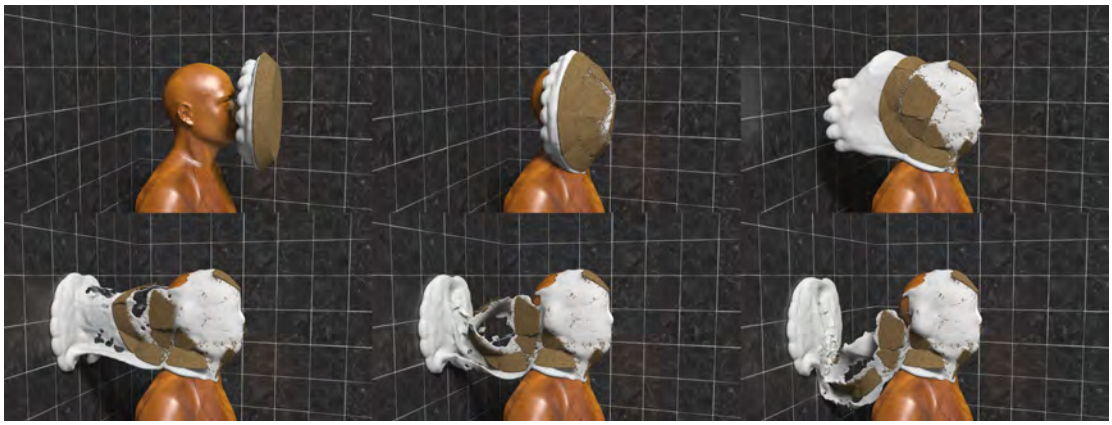

Figure 4.6: *A pie with a stiff crust and soft whipped cream is thrown at a mannequin.*

	Min/Frame	Particle #	Threads	CPU	Δx	Grid Resolution
Twisting sponge	5.3	9.1×10^5	20	3.00 GHz	0.0366	245^3
Shooting sponge	2.0	7.2×10^5	16	2.90 GHz	0.0402	175^3
Shaving foam	0.93	1.1×10^6	12	3.47 GHz	0.0019	257^3
Toothpaste	0.28	2.8×10^5	16	2.90 GHz	0.0082	$244 \times 487 \times 244$
Viennetta ice cream	1.11	1.2×10^6	12	2.67 GHz	0.0026	$385 \times 96 \times 64$
Pie	23.6	1.3×10^6	12	3.07 GHz	0.0024	333^3

Table 4.2: *Simulation performance.*

the top and bottom of the sponge are held in place as we shoot it with a kinematic bullet. The animation is in slow motion to show the detailed material response after the impact. In Figure 4.3, we simulate a stream of shaving foam hitting the ground, and compare it with real world footage. Our method captures the S-shaped buckling and merging behaviors. It also exhibits similar elastoplastic responses. In Figure 4.4, we simulate toothpaste falling onto a toothbrush. Unlike the shaving foam, Newtonian viscosity dominates material behavior. Figure 4.5 shows a simulation of manufacturing Viennetta ice cream. It captures the characteristic folding behavior. In Figure 4.6, we model a pie and throw it at a mannequin. The fracture pattern of the crust is pre-scored with weak MPM particles. The cream exhibits detailed splitting and merging behavior. For the particle-grid transfers, we used the affine Particle-In-Cell (APIC) method from Chapter 5. We found that using APIC greatly reduced positional artifacts of the pie particles.

The material parameters used in our examples are given in Table 4.1. The simulation times are shown in Table 4.2. All simulations were performed on Intel Xeon computers. All renderings were done with Mantra in Houdini. For foam, toothpaste, and Viennetta ice cream, we reconstruct surfaces from particles, and render them with subsurface scattering. The sponges were rendered as a density field.

4.5 Discussion

We found that using a Jacobi preconditioner greatly reduced simulation run times. For example, in the shooting sponge test (Figure 4.2), the Jacobi preconditioner reduces the number of CG iterations by a factor of 6.

While we have used our method successfully in simulating a variety of materials, it has some limitations. Many of these are related to the Oldroyd-B model. For example, unlike the approach in [Yue et al., 2015], our approach only handles shear thinning but not shear thickening. Therefore, the model cannot be applied to materials such as oobleck. It also does not handle material softening or hardening.

Our update rule of $\mathbf{b}_{OB}^E$ allows for inversion, which the constitutive model cannot handle. While $\mathbf{b}_{OB}^E$ should remain positive definite, we have found this to be only partially required. In particular, (4.7) involves the quantity $\text{tr}(\mathbf{b}^E)$, which we must ensure is bounded from below. If $\mathbf{b}^E$ is positive definite, then $\text{tr}(\mathbf{b}^E) > 0$. We also compute $\det(\mathbf{b}^E)^{-\frac{1}{3}}$, which is problematic if $\mathbf{b}^E$ may become singular. We avoid these problems in practice by taking advantage of the optimization-based integrator developed in Appendix A. We add a large penalty to our objective when the determinant or trace of $\mathbf{b}^E$ becomes infeasible; the line search in our optimizer then discards these configurations. While bounding the trace and determinant does not enforce definiteness in 3D, this strategy worked well in practice. Not enforcing these produces popping artifacts.

CHAPTER 5

The Affine Particle-in-Cell Method

This chapter introduces a novel mass and momentum (velocity) transfer scheme between particles and the grid [Jiang et al., 2015]. The techniques introduced here not only apply to the Material Point Method, but also work for all hybrid Lagrangian/Eulerian simulation, which is commonplace in computer graphics for fluids and other materials undergoing large deformation.

In hybrid methods, particles are used to resolve transport and topological change, while a background Eulerian grid is used for computing mechanical forces and collision responses. Particle-in-Cell (PIC) techniques, particularly the Fluid Implicit Particle (FLIP) variants have become the norm in computer graphics calculations. While these approaches have proven very powerful, they do suffer from some well known limitations. The original PIC is stable, but highly dissipative, while FLIP, designed to remove this dissipation, is more noisy and at times, unstable.

In this chapter, we present a novel technique designed to retain the stability of the original PIC, without suffering from the noise and instability of FLIP. Our primary observation is that the dissipation in the original PIC results from a loss of information when transferring between grid and particle representations. We prevent this loss of information by augmenting each particle with a locally affine, rather than locally constant, description of the velocity. We show that this not only stably removes the dissipation of PIC, but that it also enables exact conservation of angular momentum across the transfers between particles and grid.

5.1 Background

We first review some existing hybrid particle/grid methods.

Hybrid particle/grid: Several works couple SPH with grid-based techniques [Losasso et al., 2008; Hong et al., 2008a; Lee et al., 2009; Gao et al., 2009; Zhu et al., 2010a; Raveendran et al., 2011]. Sin et al. [2009]couple particles with a Voronoi grid-based pressure projection. Feldman et al. [2003] simulate explosions with a particle-based advection and grid-based pressure solve. Chentanez and Muller [2010, 2014] couple Lagrangian particles with shallow water and semi-Lagrangian techniques to adapt level of detail and use particle reseeding for sub-cell detail [Chentanez and Muller, 2011].

PIC/FLIP: Foster and Metaxas [1996] first introduced PIC techniques to computer graphics with liquid simulation. Zhu and Bridson [2005] popularized the now widely-used linear combination of FLIP and PIC. Zhu and Bridson [2005] developed a number of extensions to, including improved treatment of boundary conditions in irregular domains and coupling with rigid bodies [Batty et al., 2007], viscosity treatment [Batty and Bridson, 2008a], discontinuous-Galerkin-based adaptivity [Edwards and Bridson, 2014], multiphase flow [Boyd and Bridson, 2012], and higher-order accuracy [Edwards and Bridson, 2012]. Cornelis et al. [2014] couple high-resolution FLIP with a low-resolution implicit Smoothed Particle Hydrodynamics (SPH) from [Ihmsen et al., 2013]. Gerszewski and Bargteil [2013] use mass-full FLIP with a unilateral incompressibility constraint to resolve large-scale splashing liquids. Narain et al. [2013] also use FLIP techniques for the simulation of sand dynamics. Stomakhin et al. [2013] use MPM to simulate snow and melting/freezing [Stomakhin et al., 2014].

Level Sets: Many other graphics approaches utilize similar hybrid particle/grid data structures, particularly for resolving free-surface flows. Enright et al. [2002] use Lagrangian marker particles to improve the accuracy of the level set method for free-surface flows with the Particle Level Set Method (PLS). Mihalef et al. [2007] take a sim-

ilar approach but concentrate particles directly on the zero isocontour of the level set. [Enright et al., 2002] couple Lagrangian particles with the Particle Level Set Method to simulate compressible bubbles in incompressible flow [Patkar et al., 2013]. Kim et al. [2006a] explore further use of the escaped particles from [Enright et al., 2002]. Song et al. [2009] apply the Constrained Interpolation Profile (CIP) [Yabe et al., 2001] approach with [Enright et al., 2002] by allowing particles and grid to store velocity and level set derivative information.

In a hybrid scheme, the dual representation of the material using particles and grid provides a lot of advantages, but they also create numerous difficulties. Specifically, while the hybridization allows numerical algorithms to be done in the most appropriate representation, transferring between representations creates error. In this chapter we show how that error can be minimized with minimal effort.

While our approach will apply to a wide range of continuum phenomena, for simplicity, first consider fluid simulation. Here, pressure and viscosity updates are best done on an Eulerian grid while advection is best done with Lagrangian particles. The first and simplest method of this type is PIC [Harlow, 1964; Harlow and Welch, 1965]. While this method is remarkably effective and simple to implement, it suffers from significant dissipation (viscous appearance) due to frequent particle/grid transfers. Dissipation is addressed in the FLIP method [Brackbill and Ruppel, 1986; Brackbill et al., 1988]. The main idea is to transfer increments of velocities and displacements from grid to particles, rather than directly interpolating from the grid. Intuitively, if there is only a small offset, only a small correction will be made, typically reducing the dissipation. Unfortunately, there are other errors aside from dissipation inherent to hybrid Lagrangian/Eulerian material representations.

Specifically, the mismatch in particle and grid degrees of freedom leads to a loss of information. Since there are often more particles than grid nodes, some particle modes are not seen by the grid and get no physical response. This is the so-called “ringing

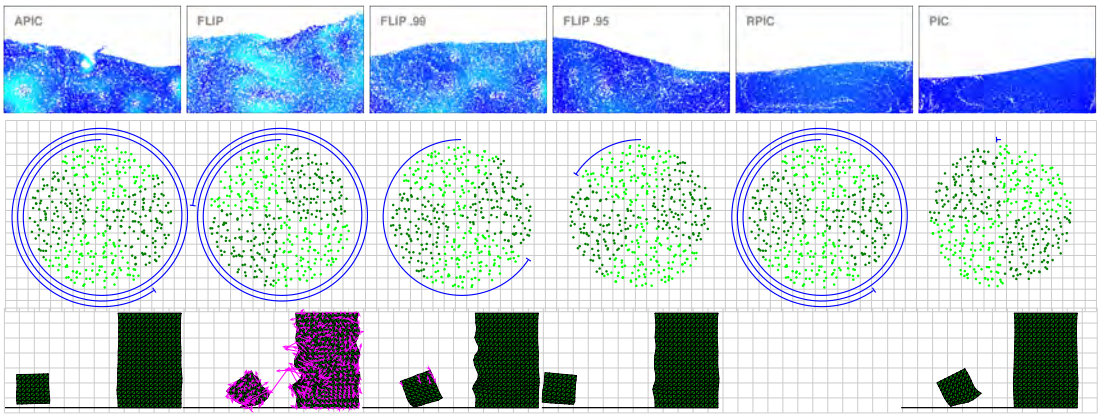

Figure 5.1: *For illustration, we compare performance with some simple 2D examples. The top row compares the methods in a dam break, free surface test. Note that APIC preserves more vorticity than even pure FLIP, while also remaining less noisy. The second row illustrates the angular momentum conservation properties of the methods. The blue spiral indicates how far the circle has rotated. The bottom row illustrates the ringing instability. Note that for pure FLIP, the velocities are large on particle but zero when transferred to grid.*

instability" (Figure 5.1) which was first-discovered in PIC [Brackbill, 1988] but is even more-pronounced in FLIP [Love and Sulsky, 2006]. Intuitively, this problem is worse in FLIP, because in PIC, particle-to-grid transfer followed by grid-to-particle transfer is a true filtering of the instability. However, while PIC forces all information through the grid, FLIP preserves some particle information, which allows the instability to persist and grow unpredictably over multiple time steps. This might lead one to believe that the ringing instability should not exist for PIC; however, it does to a lesser extent since movement of particles re-creates the instability in the next time step. In graphics simulations these instabilities lead to practical particle positional artifacts such as noise, clumping, and volume change.

A particularly problematic artifact of the dissipation with the traditional PIC approach is loss of angular momentum. The standard PIC transfer from grid to particles dissipates a significant amount of angular momentum, which leads to serious rotational artifacts (Figure 5.1). Objects in free fall disturbingly stop rotating as if under the ac-

tion of a viscous fluid drag. While FLIP was developed to reduce the dissipation of PIC, it also greatly improves the angular momentum conservation. However, FLIP will only guarantee exact conservation of angular momentum with the use of a nondiagonal (consistent) mass matrix, which is not possible in practice since the consistent mass matrix can be singular for some configurations of the particles [Love and Sulsky, 2006]. This can be remedied in practice by using an effective mass matrix equal to a weighted average of a lumped mass matrix and the consistent mass matrix, and while this does not perfectly conserve angular momentum, it is still a vast improvement over the original PIC [Love and Sulsky, 2006].

In graphics Zhu and Bridson [2005] advocate blending between pure PIC and FLIP to stabilize the simulator. While this does produce more stable behavior, it re-introduces dissipation and may require manual tuning of the blend weights on a case-by-case basis. The problem is particularly bad for thin sheets of fluid. This has been addressed in a number of ways including increasing resolution and adaptivity [Hong et al., 2008b, 2009; Ando and Tsuruno, 2011; Ando et al., 2012, 2013]. For similar reasons, Um et al. [2014] develop a sub-grid-cell corrective forcing procedure to ensure accurate particle distributions over time with FLIP and Edwards and Bridson [2012] add a regularization term to diminish particle noise not corrected by the grid.

The current state leaves us with the difficult choice for every simulation we run: (1) bias our simulation toward PIC, effectively avoiding instability at the expense of dissipation, or (2) bias our simulation toward FLIP, getting more lively simulations at the expense of noise and possible unstable behavior. In this chapter we present a third option. In particular, we control noise by keeping the pure filter property of PIC, but minimizing information loss by enriching each particle with a 3×3 matrix giving locally affine (rather than locally constant) description of the flow. Our Affine Particle-In-Cell (APIC) method effectively reduces dissipation, preserves angular momentum and prevents instabilities. Furthermore, we demonstrate that the method is applicable

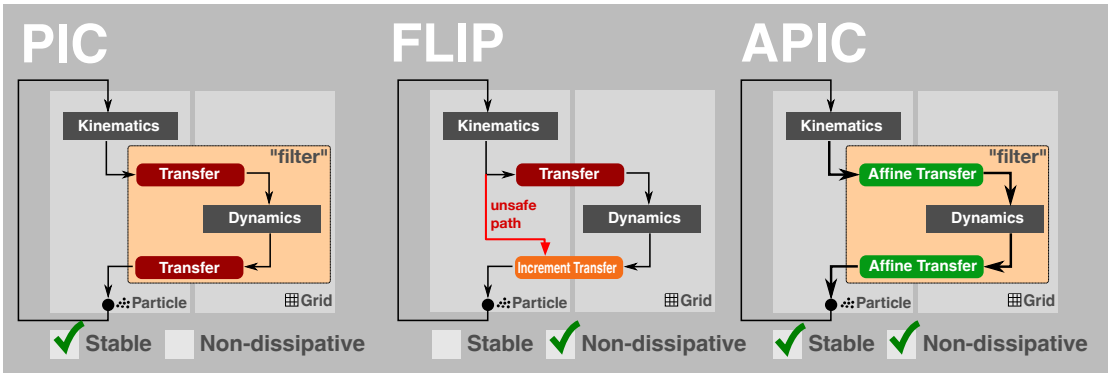

Figure 5.2: *The basic dataflow of various hybrid particle/grid simulation techniques. Both our method and PIC gain stability by using a true filtering transfer.*

to both incompressible liquids and MPM simulations [Sulsky et al., 1995].

5.2 Method Outline

A Lagrangian/Eulerian hybrid simulation time step follows a similar pattern regardless of whether one is simulating fluids with incompressible FLIP or solids with the MPM. Abstractly, kinematic steps are done on particles and dynamic steps are done on the grid. The exact form of those steps may be different with each phenomenon, and one can see examples of a canonical fluid loop in [Zhu and Bridson, 2005] or MPM loop in [Stomakhin et al., 2013]. As such the basic timestep loop for PIC, FLIP and our method is shown in Figure 5.2. The only difference between the methods is how the transfer between grid and particles is done. This difference is the focus of this chapter.

PIC is the canonical technique for coupling, and its diagram clearly shows that all data flows through the grid, and in fact the transfer from particle to grid is a pre-filter to the grid dynamics. By contrast, even though FLIP has the same pre-filter when going from particles to the grid, it introduces an additional data path directly from the original particle state. The advantage is less dissipation, but the disadvantage is an unsafe path that can lead to instability. We will show that this path is not the only way to reduce

dissipation. In fact, we show that we can obtain low-dissipation simulations while still maintaining the safety of the PIC filtering scheme.

The key observation is that normally a single particle receives data from multiple grid points, but it is typically forced to reduce those influences to a single constant value, leading to loss of information (e.g., dissipation). In Section 5.3 we develop two approaches for enriching particles to avoid this loss. Rigid Particle-in-cell (RPIC) is introduced in Section 5.3.2, and it augments each particle with the angular momentum lost in the grid to particle transfer. Unfortunately this is insufficient because shearing modes are still lost. This leads to our final method Affine Particle-in-cell (APIC) in which particles are endowed with a full affine representation of the local grid data, which we discuss in Section 5.3.3.

With our transfer technique developed, we show how to apply it to fluids in Section 5.4. We demonstrate this method on a range of interesting materials in Section 5.5.

5.3 Particle-grid transfers

5.3.1 PIC

The standard PIC routine stores mass m_p, position $\boldsymbol{x}_p^n$, and velocity $\boldsymbol{v}_p^n$. Note that m_p lacks a time n superscript because it is never changed to ensure mass conservation. Each time step begins with a transfer of mass and momentum from particles to a collocated grid according to

$$m_i^n = \sum_p w_{ip}^n m_p, \quad m_i^n \boldsymbol{v}_i^n = \sum_p w_{ip}^n m_p \boldsymbol{v}_p^n, \tag{5.1}$$

where $w_{ip}^n = N(\boldsymbol{x}_p^n - \boldsymbol{x}_i)$ are our interpolation weights and $\boldsymbol{x}_i$ denote the regular Cartesian grid node locations. With mass and momentum on the grid, we apply forces

to the velocities in a grid-based update, $\boldsymbol{v}_i^n \rightarrow \tilde{\boldsymbol{v}}_i^{n+1}$. Note that we have used $\tilde{\boldsymbol{v}}_i^{n+1}$ rather than $\boldsymbol{v}_i^{n+1}$ to distinguish them from $\boldsymbol{v}_i^n$ at the next time step. Finally, we interpolate the velocity back to particles with

$$\boldsymbol{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1}. \tag{5.2}$$

A major problem with PIC is that it severely damps rotational motion. We can get some insight into this by considering the angular momentum conservation properties in the transfers. Note that linear momentum is conserved by both transfers. The total angular momentum over the particles and grid at time t^n are given by

$$\mathbf{L}_{tot}^{P,n} = \sum_p \boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n \qquad \mathbf{L}_{tot}^{G,n} = \sum_i \boldsymbol{x}_i \times m_i^n \boldsymbol{v}_i^n. \tag{5.3}$$

While the transfer from particles to the grid conserves angular momentum, the transfer from the grid back to particles does not. This loss of angular momentum manifests as rotational motion damping (Figure 5.1).

5.3.2 Rigid Particle-In-Cell (RPIC)

In our efforts to reduce the information loss when transferring from particles to grid and vice versa, we first develop modifications to the original PIC transfer designed to facilitate conservation of angular momentum in the grid to particle transfer. Consider the case of a single particle. PIC typically transfers information to multiple grid locations since w_{ip}^n is generally nonzero for a few grid nodes at a time. Thus, even for a single particle of material, its corresponding representation on the grid is capable of storing angular momentum (by virtue of consisting of multiple grid nodes). However, one particle is incapable of representing angular momentum. Therefore, to improve the compatibility of the particle representation with the grid representation, we can addi-

tionally store a sample of local angular momentum $\mathbf{L}_p^n$ on each particle. This way, even in the case of one particle, we can prevent the loss of angular momentum when transferring from grid to particle. A similar idea was used recently by Muller et al. [2015] who augment SPH particles with a sample of angular momentum. However, because their method is SPH-based, they do not need to define particle-grid transfers.

The angular momentum that would normally be lost in the transfer from grid to particles is

$$\mathbf{L}_p^{n+1} = \sum_i w_{ip}^n (\boldsymbol{x}_i - \boldsymbol{x}_p^n) \times m_p \tilde{\boldsymbol{v}}_i^{n+1}. \tag{5.4}$$

With this definition, the total angular momentum on particles becomes

$$\mathbf{L}_{tot}^{P,n} = \sum_p (\boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \mathbf{L}_p^n). \tag{5.5}$$

Also, with this definition, the transfer from grid to particles trivially conserves angular momentum.

Next, we must define the transfer from particles to the grid. If we consider the particles to be rigid bodies with inertia tensors $\boldsymbol{K}_p^n$, then we can define the angular velocity $\boldsymbol{\omega}_p^n = (\boldsymbol{K}_p^n)^{-1}\mathbf{L}_p^n$. The rigid body's local velocity at a grid node is $\boldsymbol{v}_p^n + \boldsymbol{\omega}_p^n \times (\boldsymbol{x}_i - \boldsymbol{x}_p^n)$, which suggests the natural transfer

$$m_i^n \boldsymbol{v}_i^n = \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + ((\boldsymbol{K}_p^n)^{-1}\mathbf{L}_p^n) \times (\boldsymbol{x}_i - \boldsymbol{x}_p^n)), \tag{5.6}$$

One may imagine this transfer as distributing the masses $w_{ip}^n m_p$ from the rigid body to the grid node i. This suggests using

$$\boldsymbol{K}_p^n = \sum_j w_{jp}^n m_p (\boldsymbol{x}_j - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_j - \boldsymbol{x}_p^n)^{*T} \tag{5.7}$$

for the rigid body's inertia tensor, where $\boldsymbol{v}^*$ is the cross-product matrix associated with

vector $\boldsymbol{v}$. Performing the transfer from particles to grid in this way conserves angular momentum, as we prove in Appendix C.

5.3.3 Affine Particle-In-Cell (APIC)

While the piecewise rigid formulation corrects rotational artifacts arising from loss of angular momentum in PIC, it still damps out nonrigid motions such as shearing. We can extend the idea of enriching our velocity representation to handle shearing modes by idealizing the velocity as locally affine on each particle. This requires the additional storage of a matrix $\mathbf{C}_p$, and the local velocity represented by a particle at the grid node $\boldsymbol{x}_i$ is then $\boldsymbol{v}_p^n + \mathbf{C}_p^n(\boldsymbol{x}_i - \boldsymbol{x}_p^n)$. This can be used to define a transfer from particles to grid as in the piecewise rigid case. However, uniquely defining the nine components of $\mathbf{C}_p^n$ from the three components of $\mathbf{L}_p^n$ as in the piecewise rigid case is not possible, which complicates the process of deriving the transfer from grid to particles.

An important feature of the piecewise rigid formulation is that translational and rotational velocity fields are transferred exactly from particles to the grid and vice versa. Rather than explicitly trying to conserve angular momentum in the transfer from grid to particles, we seek to preserve affine velocity fields across both transfers. However, we show that a simple solution derived from the preservation of affine velocity fields also conserves angular momentum (Appendix C).

The transfer from particles to grid is motivated analogously to the piecewise rigid case and is of the form

$$m_i^n \boldsymbol{v}_i^n = \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_p^n)), \tag{5.8}$$

where $\mathbf{C}_p^n = \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1}$ and

$$\mathbf{D}_p^n = \sum_i w_{ip}^n(\boldsymbol{x}_i - \boldsymbol{x}_p^n)(\boldsymbol{x}_i - \boldsymbol{x}_p^n)^T \tag{5.9}$$

is analogous to an inertia tensor and is derived by preserving affine motion during the transfers. The corresponding transfer from the grid back to particles is

$$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1}(\boldsymbol{x}_i - \boldsymbol{x}_p^n)^T. \tag{5.10}$$

To discuss the angular momentum conservation properties of the transfer, we must first define angular momentum over the new particle state. A natural definition is the angular momentum on the grid after the transfer in (5.8). This takes the form

$$\mathbf{L}_{tot}^{P,n} = \sum_p m_p(\boldsymbol{x}_p^n \times \boldsymbol{v}_p^n + (\mathbf{B}_p^n)^T : \boldsymbol{\epsilon}), \tag{5.11}$$

where $\boldsymbol{\epsilon}$ is the permutation tensor (Appendix C). Note that the skew-symmetric component of $\mathbf{B}_p^n$ contains all of the angular momentum information. In this way, it is analogous to $\mathbf{L}_p^n = \boldsymbol{K}_p^n\boldsymbol{\omega}_p^n$, which combined with $\mathbf{B}_p^n = \mathbf{C}_p^n\mathbf{D}_p^n$ illustrates that $\mathbf{D}_p^n$ is analogous to the inertia tensor. Using this definition, conservation during transfer from particles to grid is automatic. However, conservation during the transfer from grid to particle can also be shown (Appendix C).

Note that despite these similarities (5.9) this is not quite the same as (5.7), and $\mathbf{D}_p^n$ does not strictly speaking have the properties of an inertia tensor. Conveniently, $\mathbf{D}_p^n$ takes on a surprisingly simple form in the case of the quadratic ($\mathbf{D}_p^n = \frac{1}{4}\Delta x^2\mathbf{I}$) and cubic ($\mathbf{D}_p^n = \frac{1}{3}\Delta x^2\mathbf{I}$) interpolation stencils commonly used for MPM. Note that for these interpolating stencils, multiplying by $(\mathbf{D}_p^n)^{-1}$ amounts to a constant scaling factor. For trilinear interpolation, a complication arises since $\mathbf{D}_p^n$ may be singular if a particle lies on a grid facet (node, edge, or face). However, in the special case of a trilinear stencil,

we have the convenient identity $w_{ip}^n(\mathbf{D}_p^n)^{-1}(\boldsymbol{x}_i - \boldsymbol{x}_p^n) = \nabla w_{ip}^n$, which allows us to avoid this numerical difficulty entirely since (5.8) can be readily evaluated without forming $(\mathbf{D}_p^n)^{-1}$.

5.4 Fluids

To apply these ideas to MAC-based fluid simulations, we formulate a set of transfers between particles and MAC faces. Specifically, we transfer from particles to faces using

$$m_{ai}^n = \sum_p m_p w_{aip}^n$$
$$m_{ai}^n v_{ai}^n = \sum_p m_p w_{aip}^n (\mathbf{e}_a^T \boldsymbol{v}_p^n + (\mathbf{c}_{pa}^n)^T (\boldsymbol{x}_{ai} - \boldsymbol{x}_p^n))$$

and from faces to particles using

$$\boldsymbol{v}_p^{n+1} = \sum_{a,i} w_{aip}^n \tilde{v}_{ai}^{n+1} \mathbf{e}_a \quad \text{and} \quad \mathbf{c}_{pa}^{n+1} = \sum_i \nabla w_{aip}^n \tilde{v}_{ai}^{n+1}.$$

Here, a represents an x, y, or z face direction, and $\boldsymbol{x}_{ai}$ is the location of a MAC face associated with direction a. The weights are $w_{aip}^n = N(\boldsymbol{x}_{ai} - \boldsymbol{x}_p^n)$, where $N(\boldsymbol{x})$ is chosen to be the trilinear interpolation kernel. m_{ai}^n and v_{ai}^n are the mass and velocity component on the MAC face, and $\mathbf{c}_{pa}$ is a vector per axis, notably requiring the same amount of storage as the collocated case. Incompressibility is enforced in the standard way [Bridson, 2008].

5.5 Simulation results

Free-surface flow: The most common graphics application of PIC and FLIP is free-surface incompressible flow (Figure 5.3). We compare APIC with pure FLIP, PIC and

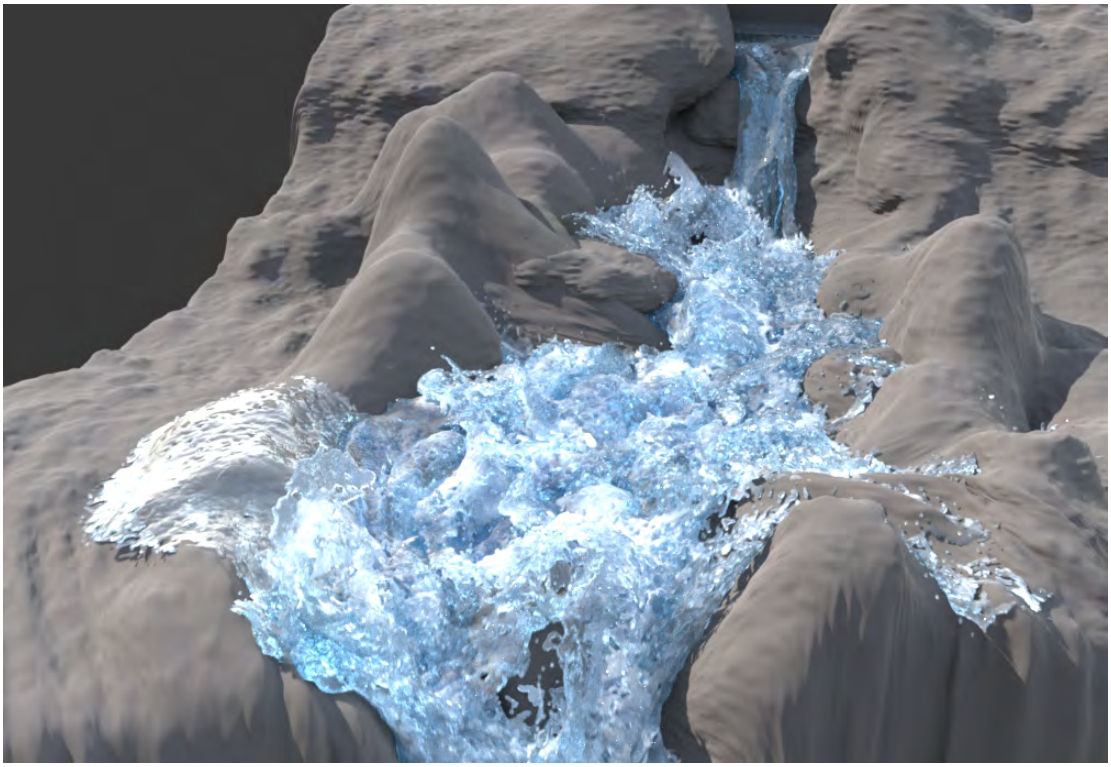

Figure 5.3: *APIC resolves the complex free-surface dynamics of a rushing river on a rugged terrain.*

Figure 5.4: *Here we compare the methods with a fountain simulation of free-surface flow. The top row shows that while APIC and FLIP are the least dissipative in the initial stages, the FLIP surface is already displaying a noisy leading edge relative to the more smoothly resolved APIC. The bottom row shows that the entire surface of the fountain becomes noisy with FLIP in the later stages.*

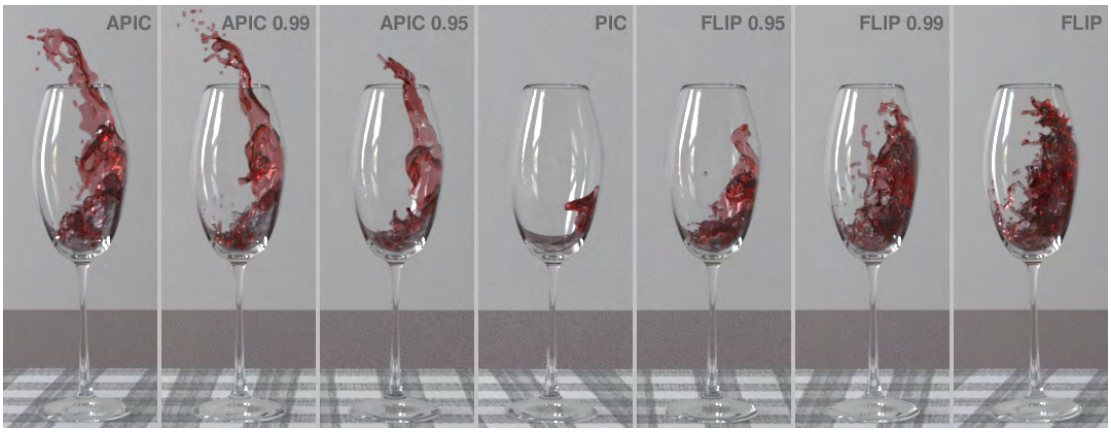

Figure 5.5: *APIC/PIC blends yield more energetic and more stable behavior than FLIP/PIC blends in a wine pouring example. APIC/PIC blends are achieved analogously to FLIP/PIC in that it is a scaling of the particle affine matrices.*

FLIP/PIC blends in a number of free-surface calculations. Figure 5.4 shows a comparison with a fountain example. The top row (earlier time) shows that while APIC and FLIP start as the least dissipative, the FLIP surface is already displaying a noisy leading edge relative to the more smoothly resolved APIC. PIC is the most damped, and any amount of PIC/FLIP blending leads to a damped leading edge position. The bottom row (later time) shows that FLIP simulates a noisy surface. which is still visible on FLIP/PIC blends. In Figure 5.5 we demonstrate the behavior of APIC/PIC blends compared to FLIP/PIC. We note that unlike with FLIP, pure APIC is clearly superior to APIC/PIC blends. Note that APIC produces more smooth and stable wine surfaces than FLIP/PIC blends while simultaneously resolving more energetic splashing behavior than even pure FLIP.

Granular materials: In Figure 5.6 we compare APIC with FLIP and FLIP/PIC blends. We model the sand as a granular material using the constitutive model from [Stomakhin et al., 2013] and Chapter 3. We use an inlet condition at the top of the slide to induce a mixing flow. The dynamics demonstrate the noise of pure FLIP and the dissipation of PIC and FLIP/PIC blends. APIC is able to retain the stability of PIC and FLIP/PIC blends without the excessive dissipation. We show another comparison with granular

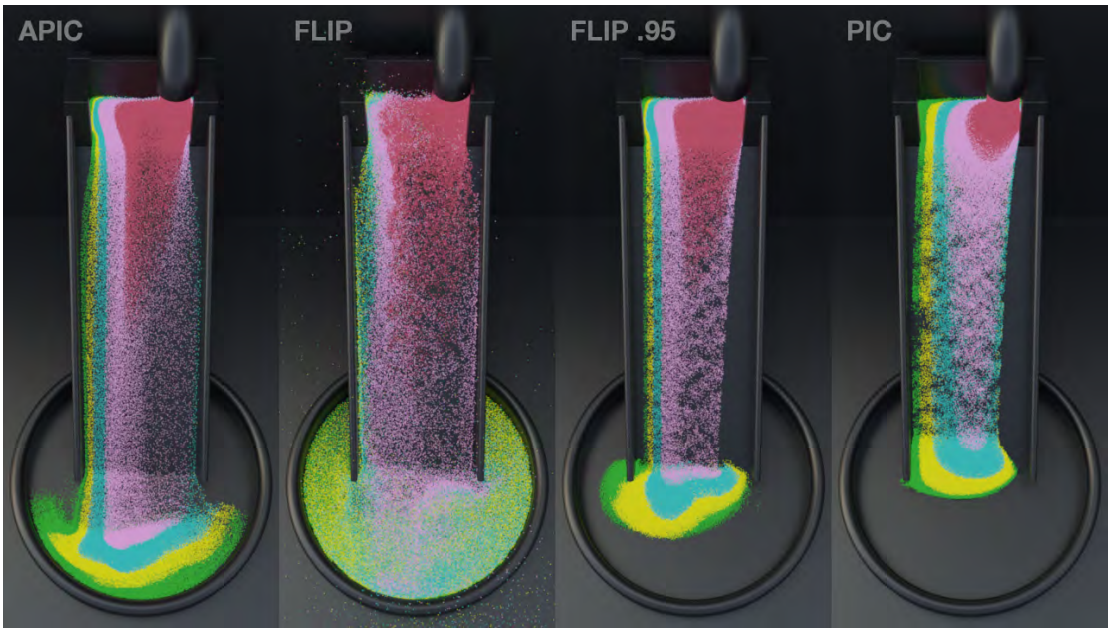

Figure 5.6: *We compare APIC with PIC, FLIP and FLIP/PIC blends for an MPM simulation of granular materials. Notice PIC and FLIP/PIC blend are stable but exhibit overly viscous behavior, while pure FLIP is unstable and noisy as evidenced by stray particles and excessive mixing. APIC however is both stable and nearly dissipation free.*

materials in Figure 5.7. Here, FLIP again exhibits overly noisy behavior. The dissipation in the PIC method causes the sand to bunch together giving it a wet look. While the FLIP/PIC blend is more stable than pure FLIP, it also suffers from the clumping, wet look of PIC. However, APIC stably resolves the dynamics of a fine powdery sand. In Figure 5.8 we show that APIC is better able to retain angular momentum without the dispersive behavior of FLIP.

Elastic solids: A coupling example can be seen in Figure 3.1. Here we use a traditional MPM discretization of the elastoplastic constitutive model from [Bargteil et al., 2007] to simulate frozen yogurt. We couple this with an elastic cloth using the Lagrangian force model. The cloth is modeled using a standard mass-spring energy. In Figure 3.2 we show a comparison using mesh-based cubes with a Lagrangian finite element constitutive model. Here, APIC retains angular momentum and energetic behavior better than PIC, FLIP, and FLIP/PIC blends. FLIP and FLIP/PIC blends produce ringing dur-

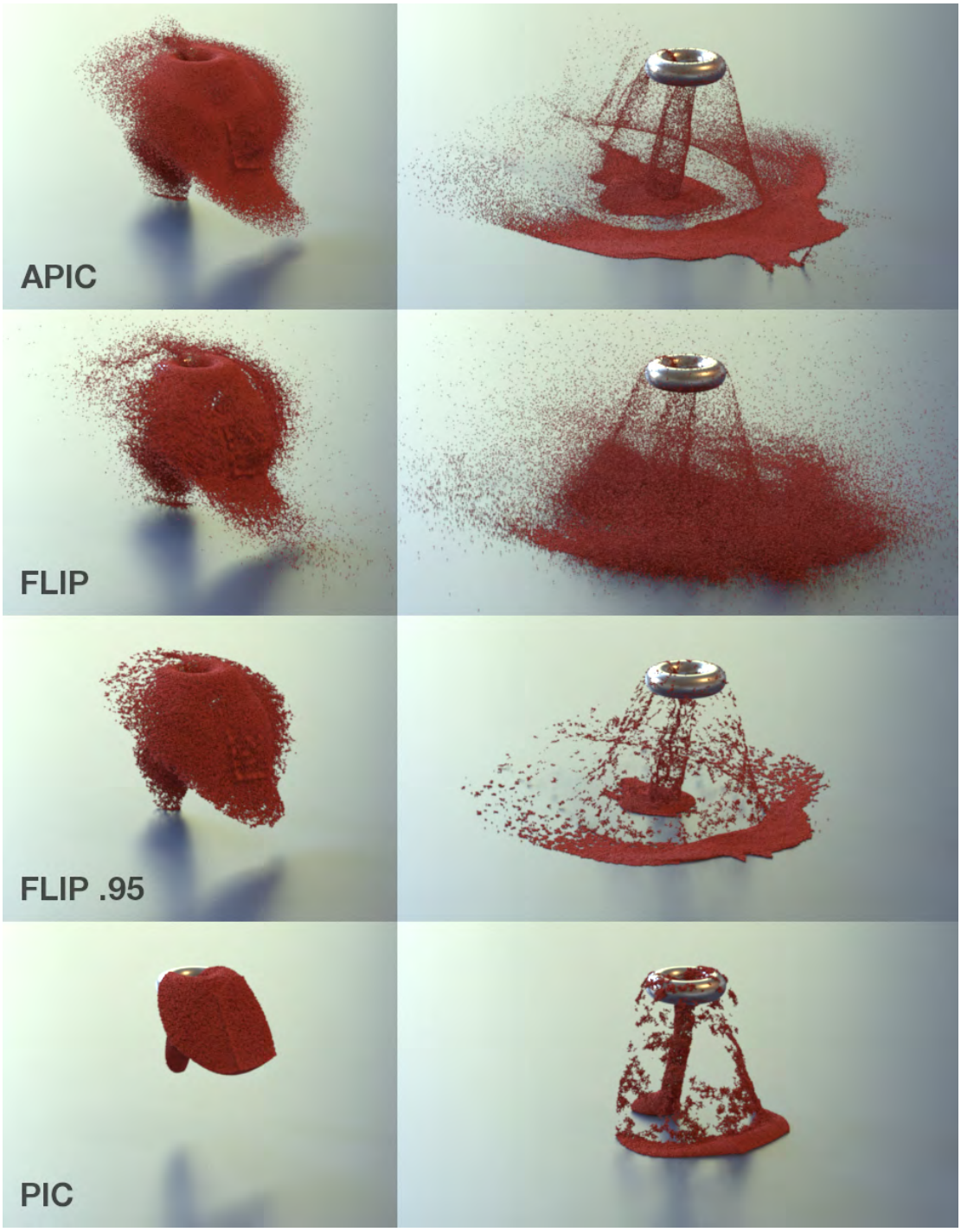

Figure 5.7: *We again compare against FLIP and APIC with a granular material example. Here, a red cube of sand is dropped onto a torus causing it to exhibit interesting flow patterns. APIC appears more like a fine powder—while FLIP suffers from excessive noise and instability. The dissipative nature of PIC causes the sand to clump together, giving it a wet look that also plagues the FLIP/PIC blend.*

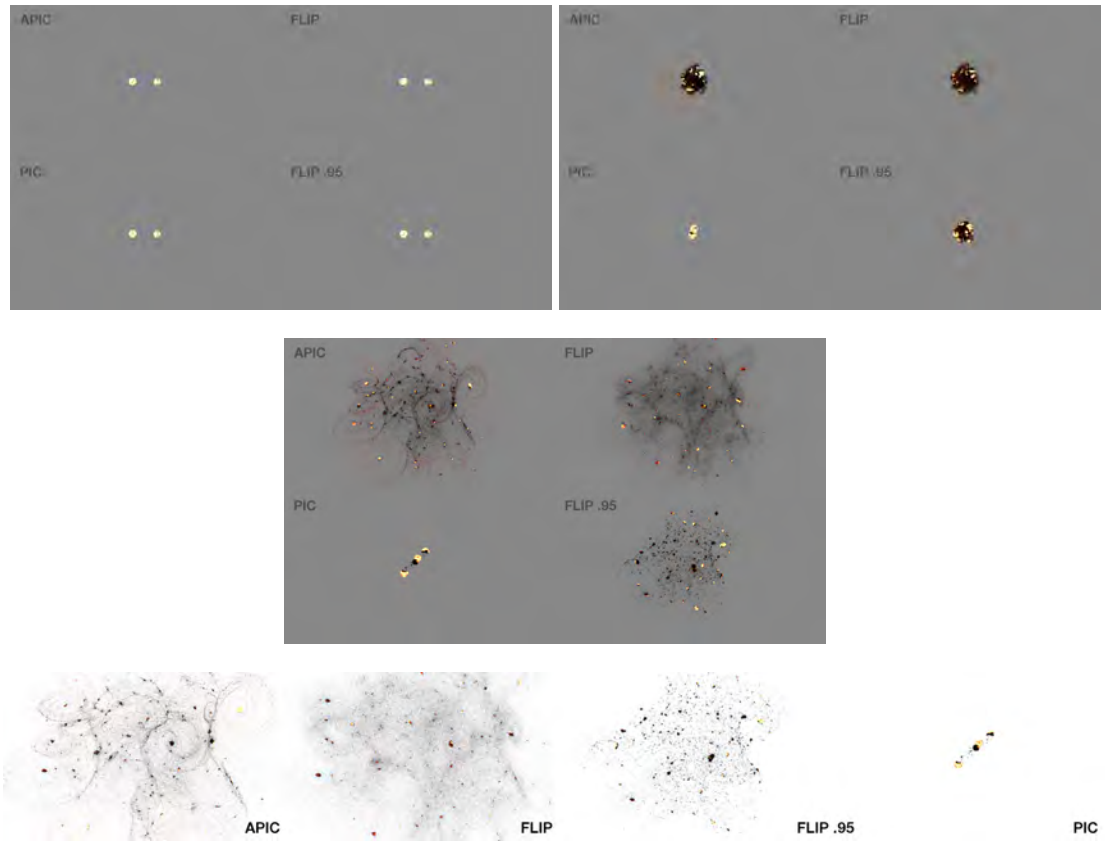

Figure 5.8: *We compare APIC with FLIP and PIC in a high energy collision example. APIC resolves interesting spiral shedding behavior that the other methods cannot.*

ing the collisions between the cube and the glass plates and flexible block. For the underlying method of coupling the cloth and yogurt, see Section 3.3.

Lava: We demonstrate the benefits of APIC for lava flows using the model from [Stomakhin et al., 2014]. In Figure 5.9 we show interesting lava flows over a rugged terrain. We directly compare APIC with FLIP and FLIP/PIC blends in Figure 5.10. In this example, a spout pours lava with cooler, rockier properties near the outer circumference into free-fall. APIC resolves an interesting periodic flow, while pure FLIP goes unstable, with hotter interior particles noisily exploding outwards. FLIP/PIC blends stabilize this behavior, but do not produce flows as detailed as with APIC.

Figure 5.9: *Here we demonstrate the performance of APIC with an elastoplastic model for lava flow.*

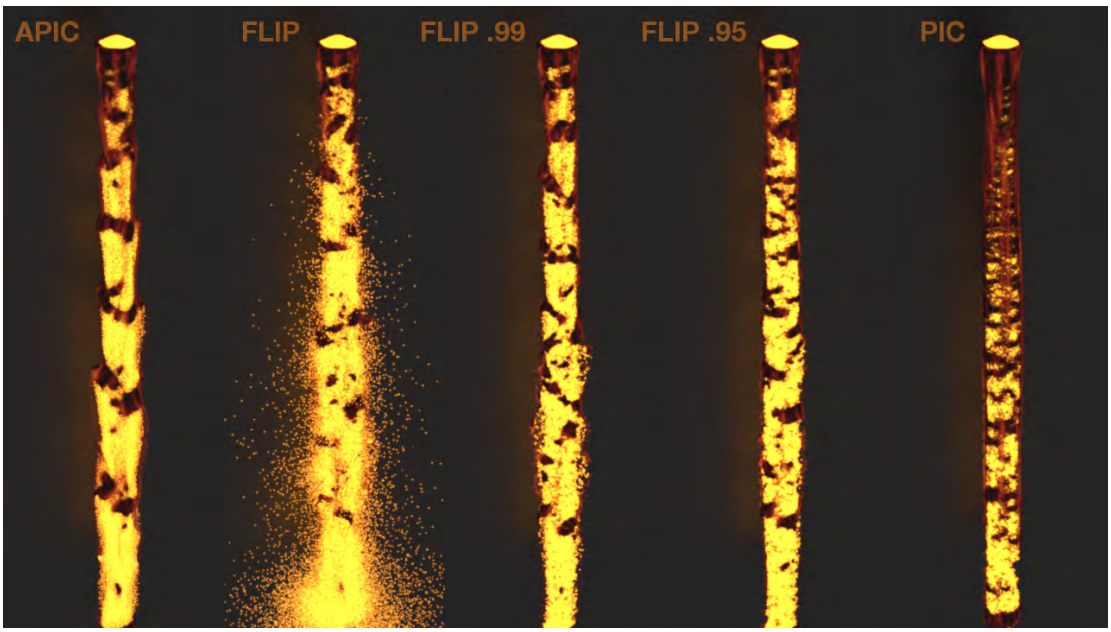

Figure 5.10: *We compare APIC with FLIP and PIC during a lava in free-fall example. Pure FLIP is unstable which leads to particles exploding out from the interior. FLIP/PIC blends do not suffer from this, but they cannot resolve the detailed flows shown in APIC.*

Method	FLIP			PIC	APIC		
Mixture	1	0.99	0.95	0	0.95	0.99	1
Wine	1.2	1.1	1.1	1.0	1.0	1.1	1.1
River		1.0	1.1	1.0			1.2
Collision	9.5	3.2	1.5	1.0			5.0
Lava Free-fall	2.5	1.1	0.9	1.0			1.3
Water fountain	1.1	1.2	1.5	1.0			1.0
Sand incline	4.0	1.9	1.5	1.0			2.2

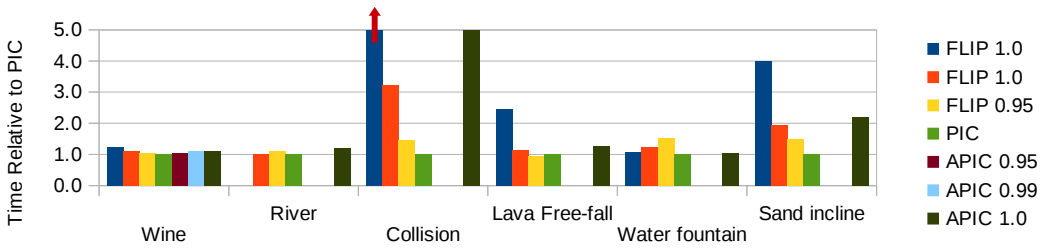

Figure 5.11: *We use relative timing to abstract performance impact from our particular simulator. Typically simulation performance is correlated to CFL, thus unstable and lively simulations take longer.*

5.6 Discussion

Although we eliminate nearly all of the artificial dissipation of pure PIC, our method neither improves nor exacerbates the ringing instability. This is quite unlike FLIP, however, which avoids dissipation at the cost of losing stability.

A minor disadvantage of our approach is the need to store an extra matrix per particle and perform a few extra operations during the transfers. In practice, we have found the extra storage and transfer cost to be negligible as runtime costs are typically dominated by the magnitude of the velocities and our CFL. PIC tends to be fastest, since it damps out motion and has the smallest velocities. On the other hand, FLIP tends to be slowest due to its instability and consequent larger velocities. In particular, we see that the wine timings in Figure 5.11 are fairly uniform across all methods, because the maximum velocity is similar. Similarly, unstable FLIP simulations like the high energy collision tend to be very slow.

It is interesting to note that a stable FLIP blend often contains artifacts that are desirable,

especially in the case of liquids and wet sand. FLIP in some ways is akin to forcing functions that are required to make grid-based smoke solvers visually interesting. We assert, however, that if such instabilities are desirable, the artist would prefer to create them in a way they desire rather than have them uncontrollably imposed by the method.

CHAPTER 6

Simulating Melting and Solidification

In this chapter, we introduce a novel Material Point Method (MPM) for heat transport, melting and solidifying materials. This brings a wider range of material behaviors into reach of the already versatile MPM. This is in contrast to best-of-breed fluid, solid, or rigid-body solvers that are difficult to adapt to a wide range of materials. Extending the material point method requires several contributions. We introduce a dilational/deviatoric splitting of the constitutive model and show that an implicit treatment of the Eulerian evolution of the dilational part can be used to simulate arbitrarily incompressible materials. Furthermore, we show that this treatment reduces to a parabolic equation for moderate compressibility and an elliptic, Chorin-style projection at the incompressible limit. Since projections are naturally done on Marker-And-Cell (MAC) grids, we devise a staggered-grid MPM method. Lastly, to generate varying material parameters, we adapt a heat-equation solver to a material point framework.

6.1 Background

From the process of lava solidifying into *pāhoehoe* to advertisements showing molten chocolate solidified over ice cream, materials undergoing phase transitions are both ubiquitous and complex. These transitional dynamics are some of the most compelling natural phenomena. However, visual simulation of these effects remains a challenging open problem. The difficulty lies in achieving robust, accurate and efficient simulation

of a wide variety of material behaviors without requiring overly complex implementations.

Phase transitions and multiple material interactions typically involve large deformation and topological changes. Thus a common approach is to use a modified fluid solver, which works well for viscous Newtonian fluids or even moderately viscoplastic flows. However, solid and elastic material behavior is then more difficult to achieve. Alternatively, Lagrangian-mesh-based approaches naturally resolve elastic deformation, but they must be augmented with explicit collision detection, and re-meshing is required for fluid-like behaviors with topological changes. Due to the trade-offs between solid and fluid methods, many authors have considered explicit coupling between two solvers, but such approaches typically require complex implementations and have significant computational cost.

A common goal is to handle a variety of materials and material transitions without sacrificing simplicity of implementation. This motivation typically drives researchers and practitioners toward particle approaches. For example, SPH and FLIP methods are commonly augmented with an approach for computing strains required for more general elastic stress response. The key observation is that particles are a simple and extremely flexible representation for graphics. This is a central motivation in our approach to the problem.

Computing strain from world-space particle locations without the luxury of a Lagrangian mesh proves challenging. One approach is using the MPM [Sulsky et al., 1995], which augments particles with a transient Eulerian grid that is adept at computing derivatives and other quantities. However, while MPMs successfully resolve a wide range of behaviors, they do not handle arbitrarily incompressible materials. This is in contrast to incompressible FLIP [Zhu and Bridson, 2005] techniques that naturally treat liquid simulation but typically only resolve pressure or viscosity-based stresses.

In this chapter, we present a number of contributions. We show that MPM can be easily augmented with a Chorin-style projection [Chorin, 1968] technique that enables simulation of arbitrarily incompressible materials, thus providing a connection to the commonly used FLIP techniques. We achieve this with a MAC-grid-based [Harlow and Welch, 1965] MPM solver, a splitting of the stress into elastic and dilational parts, a projection-like implicit treatment of the Eulerian evolution of the dilational part, and careful attention to how quantities are rasterized and updated on the grid. Additionally, we couple a simple yet practical heat model to our material point solver, allowing us to drive material changes with temperature and phase.

Thermodynamic variation of material properties to achieve melting and solidifying effects for visual simulation was first explored in the pioneering work of Terzopoulos et al. [1991]. Since then, such explorations have remained very popular. A common requirement in such approaches is the unified treatment of a wide variety of material behaviors. While specialized techniques for single materials are relevant when discussing prior approaches, we primarily restrict the following literature discussion to papers that explicitly consider multiple materials with solidification and melting.

Particle-based melting: SPH is commonly used for modeling viscosity and pressure response in liquids and has been popular in graphics since the work of Desbrun and Gascuel [1996]. Because of its wide use, SPH has been frequently modified with more general strain computations that allow more general stress response. For example, Solenthaler et al. [2007] simply use standard SPH interpolation to create a continuous displacement field from per-particle world space positions that can be differentiated in material-space to obtain per-particle displacement gradients. Becker et al. [2009] show however that this displacement differentiation approach cannot accurately resolve material rotations, so they instead propose a shape-matching approach [Müller et al., 2005]. Chang et al. [2009] handle viscoelastic and melting flow by computing the strain using a convenient Eulerian evolution (as in [Goktekin et al., 2004]) that only requires

SPH interpolation of the velocities in world space. A number of other SPH methods have used Moving Least Squares to compute the strain. Keiser et al. [2005] and Müller et al. [2004] handle the transition from solid to fluid by including both traditional SPH-based pressure forces with elastic forces defined from an elastic potential defined via the moving least squares approximation to the deformation gradient. While moving least squares approaches do not suffer from the rotational artifacts encountered in the more straightforward methods of Solenthaler et al. [2007] and Becker et al. [2009], they are plagued by a number of failure scenarios where inversion of the associated moment matrices are not defined (e.g., co-planar and co-linear particle configurations as discussed in [Becker et al., 2009]). Paiva et al. [2009] and Paiva et al. [2006] avoid the need for strain computation altogether instead using non-Newtonian modifications of fluid viscosity to achieve complex fluid effects useful in melting and solidifying. Other notable uses of SPH for melting effects include [Stora et al., 1999] for lava flows, [Iwasaki et al., 2010] and [Lii and Wong, 2013] for melting ice and [Lenaerts and Dutre, 2009] for the treatment of porous granular materials and water.

Mesh-based melting: Lagrangian meshes have long been popular due to trivial per-element strain computation that leads to accurate elastic behavior [Teschner et al., 2004]. However, fluid and melting behaviors necessitate topological change, requiring remeshing. Bargteil et al. [2007] achieved efficient remeshing, Wojtan and Turk [2008] increased efficiency and fidelity using embedded meshes, and Wojtan et al. [2009] included the treatment of splitting. Wicke et al. [2010] introduce a dynamic local remeshing algorithm that attempts to replace as few tetrahedra as possible limiting the number of visual artifacts. Clausen et al. [2013] used tetrahedron-based remeshing to melt viscoelastic solids into fluids. Kim et al. [2006b] model ice dynamics as a thin film Stefan problem and represent ice volumes with a level set method.

Grid-based melting: Eulerian methods are natural when melting into a fluid phase. However, the challenge is then the computation of elastic strain. Goktekin et al. [2004];

Losasso et al. [2006a] use an Eulerian update rule for the strain evolution. Rasmussen et al. [2004] achieve melting effects by simply increasing viscosity in an Eulerian approach. Wojtan et al. [2007] include erosion phase change effects with a level set representation of fluids and eroding solids. Zhao et al. [2006] use a modified Eulerian lattice Boltzmann method to treat melting and flowing. Wei et al. [2003] use a cellular-automata-based simplification of the physics. Losasso et al. [2006b] couple a Lagrangian mesh representation of a solid with Eulerian representations of a fluid to treat each phase in the melting process. [Carlson et al., 2002b] also combine Lagrangian and Eulerian approaches by using particles for material advection and a MAC grid for implicit viscosity and pressure projection.

Heat and phase transitions: Heat evolution is typically achieved by solving the heat equation in the world space of the system. The local temperature of the material can then be used to modify its mechanical properties. Stora et al. [1999] varied viscosity with temperature to simulate lava flows. Terzopoulos et al. [1991]; Teschner et al. [2004]; Zhao et al. [2006]; Losasso et al. [2006a]; Iwasaki et al. [2010]; Clausen et al. [2013] model phase transition using a hard freezing temperature threshold. On the other hand, Carlson et al. [2002b]; Keiser et al. [2005]; Paiva et al. [2006]; Solenthaler et al. [2007]; Chang et al. [2009]; Paiva et al. [2009]; Dagenais et al. [2012] define a more smoothed material property range in the phase transition region, perhaps to model the latent heat. Maréchal et al. [2010]; Lii and Wong [2013] more correctly model phase transition including latent heat.

6.2 Method Overview

Goal: Our goal is to simulate a wide variety of materials, with the specific area of focus being volumetric simulation in the presence of phase change. A fully general unified simulation model is beyond the scope of our work, and such a model would need to

consider many more interactions. Researchers have considered some of these other goals with coupling [Carlson et al., 2004; Chentanez et al., 2006; Robinson-Mosher et al., 2008] and multi-material unification [Martin et al., 2010] (these citations are not exhaustive). Our focus is on heat-driven material change, in particular, because it requires handling a wide range of material behaviors and the transition within that range. We stress, however, that if a practitioner requires only one material at a time, computational efficiency might be obtained by using a specialized solver (e.g. FLIP for liquids).

MPM limitations: [Stomakhin et al., 2013] demonstrates that material point methods occupy an interesting middle ground for simulation techniques, especially elastoplastic materials undergoing fracture. By adding plasticity to the basic constitutive model energy in [Stomakhin et al., 2012], they show that a range of compressible materials (like snow) can be simulated. While incompressibility can be approached by increasing the Poisson's ratio, at some point locking can occur [Mast et al., 2012]. At that point one might decide to use a much simpler incompressible FLIP method. However, more generally speaking, numerical systems can usually be formulated using hard constraints or soft constraints. Soft constraints can vary stiffness, but at sufficiently high stiffness, hard constraint formulations become efficient and necessary. For example, stiffer mass-spring systems can approach rigidity, but practitioners usually turn to the reduced-coordinate rigid-body systems. Analogously, liquids can be simulated using equation-of-state SPH, but Incompressible SPH [Solenthaler and Pajarola, 2009] is often more efficient. Regardless, in the presence of material transition, it becomes difficult to switch different parts of the domain between hard constraints and soft constraints, so soft constraint methods are used everywhere. This serves to motivate the key idea, to bring some of the efficiency of hard-constraint incompressible FLIP methods to soft-constraint MPM techniques like [Stomakhin et al., 2013].

Contributions: Our basic approach is to combine the projection ideas present in in-

compressible FLIP with the rich constitutive material properties of MPM to get a very flexible solver. Our particular contributions are as follows:

1. We carefully model *heat in the context of MPM* by solving the heat equation on a background grid. Using the resulting temperature and phase, we can vary material properties like the Young's modulus and Poisson ratio. To solve for specifically problematic parameters that cause MPM locking, we develop a *generalized Chorin-style projection*, further requiring a *MAC-style staggered MPM formulation*.

2. We further show that a *deviatoric/dilational splitting of the constitutive model* naturally allows for this while facilitating arbitrary variation from compressible to incompressible.

3. We also show that sharp phase transitions also benefit from a deviatoric strain-based energy density function because it prevents energy gain when transitioning from fluid to solid.

In essence, our method can be seen as a combination of an incompressible FLIP solver and a material point solver, with material properties driven by temperature distribution defined by a heat solver operating in parallel.

We next proceed to explain the details of our solver. A visual diagram of our method is shown in Figure 6.1. In Section 6.3 we derive the physical equations for the mechanical evolution and heat transfer, as well as our splitting scheme. In Section 6.4 we discuss the details of our algorithm. Finally we present results in Section 6.5 and discussion in Section 6.6.

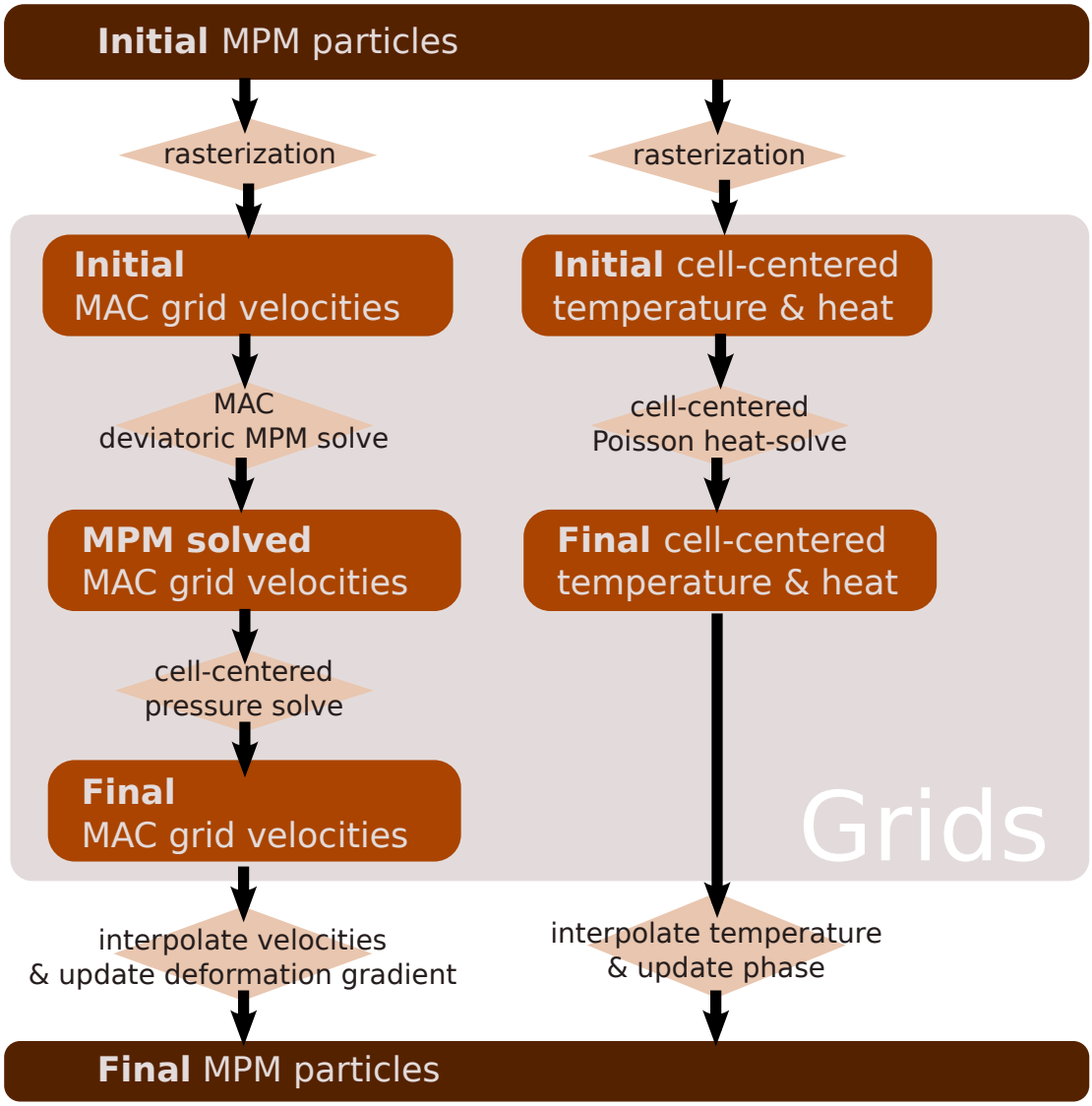

Figure 6.1: *Our method benefits from the interplay of grids and particles. In parallel with our mechanical evolution we have a thermodynamic evolution that also uses grids as a scratchpad.*

6.3 Physical Model

We describe the mapping from points in an initial material configuration $\mathbf{X}$ to their deformed state $\boldsymbol{x}$ by a transform $\boldsymbol{x} = \boldsymbol{\phi}(\mathbf{X})$. We use the notation $\boldsymbol{F} = \partial\boldsymbol{\phi}/\partial\mathbf{X}$ to describe the Jacobian (or deformation gradient) of the mapping. Material motion is governed by conservation of mass, conservation of momentum, and the elastoplastic constitutive relation

$$\frac{D\rho}{Dt} = 0, \quad \rho\frac{D\boldsymbol{v}}{Dt} = \nabla\cdot\boldsymbol{\sigma} + \rho\boldsymbol{g}, \quad \boldsymbol{\sigma} = \frac{1}{J}\frac{\partial\Psi}{\partial\boldsymbol{F}_E}\boldsymbol{F}_E^T, \tag{6.1}$$

where ρ is density, t is time, $\boldsymbol{v}$ is velocity, $\boldsymbol{\sigma}$ is the Cauchy stress, $\boldsymbol{g}$ is gravity, Ψ is the elastoplastic potential energy density, $\boldsymbol{F}_E$ is the elastic part of the deformation gradient $\boldsymbol{F}$, and $J = \det(\boldsymbol{F})$ (e.g., [Bonet and Wood, 1997]).

6.3.1 Heat flow and phase transition

The heat flow is governed by

$$\rho\frac{Du}{Dt} = -\nabla\cdot\boldsymbol{q}, \quad \boldsymbol{q} = -\kappa\nabla T, \quad c = \frac{du}{dT}, \tag{6.2}$$

where u is stored heat energy per unit mass, T is temperature, $\boldsymbol{q}$ is heat flux, κ is heat conductivity in accordance with Fourier's Law, and c is heat capacity per unit mass (e.g., [Gonzalez and Stuart, 2008]). Eliminating u and $\boldsymbol{q}$ leads to the heat equation

$$\rho c\frac{DT}{Dt} = \nabla\cdot(\kappa\nabla T). \tag{6.3}$$

This is a simplified model in that we assume no transfer between the mechanical and heat energy of the system (and hence u is a function of T only). Even so, this is the most popular approach for simulating heat transfer in graphics. Also, instead of representing

volumetric heat source terms, we use heat boundary conditions: Dirichlet or Neumann, depending on the desired behavior.

To complete our physical model, we must form a thermo-mechanical model by bringing our heat and mechanical systems together. This is accomplished by varying Ψ with temperature and phase. In particular, we use different expressions for Ψ depending on whether the material is in a solid or liquid state. It is worth noting that stable transition between two phases requires the careful treatment discussed later.

We discretize the temperature evolution in time from (6.3) as

$$T^{n+1} - T^n = \frac{\Delta t}{\rho^n c^n} \nabla \cdot (\kappa^n \nabla T^{n+1}). \tag{6.4}$$

Note, however, that this equation describes temperature evolution only within one phase. Phase transition is a separate process in the sense that it requires extra heat, so called latent heat, which cannot be observed as a temperature change. Specifically, the latent heat of fusion L of an object is the heat required to transfer it from a solid into a liquid state isothermally at the freezing point of the material (e.g., [Serway and Jewett, 2009]). Thus, the transition does not happen instantly at the freezing point, and the importance of capturing this effect is discussed in [Lii and Wong, 2013].

Some researchers mimic the effect of latent heat by expanding the temperature range in the vicinity of the freezing point and introducing separate temperatures for melting and freezing [Carlson et al., 2002b; Keiser et al., 2005; Paiva et al., 2006; Solenthaler et al., 2007; Paiva et al., 2009; Chang et al., 2009; Dagenais et al., 2012]. While this approach is sufficient for handling phase transition of a single material, it is not generally applicable to mixtures of materials with different thermal properties, since the expanded temperature ranges would not necessarily agree. We thus will follow the approach of [Maréchal et al., 2010; Lii and Wong, 2013] to accurately handle the effect latent heat in the multimaterial case. We discuss our latent heat treatment in Section 6.4.9.

In general, material parameters—e.g., μ and λ—can be defined as functions of the current temperature in addition to them being functions of the current phase; however, in practice we found that keeping them constant with T and letting $\mu = 0$ in the fluid phase was sufficient to produce visually compelling results.

6.3.2 Constitutive model

For a realistic treatment of melting and freezing, we require a suitable and well-behaved handling of plasticity and transition between liquid and solid phases of the materials. Following the multiplicative plasticity treatment of Stomakhin et al. [2013], we separate $\boldsymbol{F}$ into an elastic part $\boldsymbol{F}_E$ and a plastic part $\boldsymbol{F}_P$ so that $\boldsymbol{F} = \boldsymbol{F}_E\boldsymbol{F}_P$. With this separation, we base our constitutive model on the elastoplastic fixed co-rotational energy density function [Stomakhin et al., 2012, 2013]

$$\Psi(\boldsymbol{F}_E) = \Psi_\mu(\boldsymbol{F}_E) + \Psi_\lambda(J_E), \tag{6.5}$$

where

$$\Psi_\mu(\boldsymbol{F}_E) = \mu\|\boldsymbol{F}_E - \boldsymbol{R}_E\|_F^2, \quad \Psi_\lambda(J_E) = \frac{\lambda}{2}(J_E - 1)^2, \tag{6.6}$$

$J_E = \det(\boldsymbol{F}_E)$, and $\boldsymbol{R}_E$ is the rotation from the polar decomposition of $\boldsymbol{F}_E$. This constitutive model is known to be suitable for solids, where μ and λ are typically set from Young's modulus and Poisson's ratio of the material. Furthermore, letting $\mu = 0$ makes the energy density depend only on the local volume change and thus is suitable for liquids, both compressible and incompressible (in the $\lambda \to \infty$ limit). In fact, in this case it can be shown that the Cauchy stress is a scalar pressure. Specifically, J_E measures relative volume change, and Ψ_λ penalizes it, facilitating volume preservation.

Note however, that Ψ_μ is not completely orthogonal to Ψ_λ in the sense that it also penalizes volume change. In addition, it penalizes deviatoric strains to which Ψ_λ is oblivious. Thus, simply overriding Ψ_μ in (6.5) when changing phase is unsuitable for

freezing, as this transition would result in a sudden large increase in potential energy and produce popping artifacts. Clearly this energy increase must be avoided if freezing is to be possible.

To better understand where the energy increase comes from, consider the dilational $(J_E)^{\frac{1}{d}}\mathbf{I}$ and deviatoric $(J_E)^{-\frac{1}{d}}\boldsymbol{F}_E$ parts of $\boldsymbol{F}_E$, where d is the dimension and $\mathbf{I}$ is the identity matrix. The first source of energy is the consequence of the deviatoric component of $\boldsymbol{F}_E$. The deviatoric part is not used in Ψ_λ and would generally change quite drastically with the flow. To remedy this, we note that fluids are almost perfectly plastic with respect to deviatoric strain. We incorporate this into our model by clearing the deviatoric component from $\boldsymbol{F}_E$ immediately after it is updated by letting $\boldsymbol{F}_E \leftarrow (J_E)^{\frac{1}{d}}\mathbf{I}$ at the end of each time step in the fluid phase.

This fluid plasticity does not completely eliminate the problem, since Ψ_μ is nonzero even if $\boldsymbol{F}_E$ contains only a dilational component. To address this, we eliminate the dilational component explicitly from Ψ_μ. This is commonly done for nearly-incompressible materials [Bonet and Wood, 1997] and helps allow for arbitrarily large λ. That is, we define an alternative energy density function

$$\hat{\Psi}(\boldsymbol{F}_E) = \hat{\Psi}_\mu(\boldsymbol{F}_E) + \Psi_\lambda(J_E) \tag{6.7}$$

$$\text{where} \quad \hat{\Psi}_\mu(\boldsymbol{F}_E) = \Psi_\mu(J_E^{-\frac{1}{d}}\boldsymbol{F}_E). \tag{6.8}$$

The derivatives of Ψ_μ and Ψ_λ are as in [Stomakhin et al., 2013], and the chain rule gives us the deviatoric stress $\boldsymbol{\sigma}_\mu = \frac{1}{J}\frac{\partial \hat{\Psi}_\mu}{\partial \boldsymbol{F}_E}\boldsymbol{F}_E^T$ where, for clarity, $\frac{\partial \hat{\Psi}_\mu}{\partial \boldsymbol{F}_E}(\boldsymbol{F}_E)$ is an evaluation of a function at $\boldsymbol{F}_E$. See Appendix D for details related to the derivative terms arising from the chain rule.

6.3.3 Pressure Splitting

The model as stated would handle some material variation, but locking could occur in highly incompressible materials. This section shows how to prevent locking by transforming our solid model into a more fluid-like form whose resulting discretization will be much more efficient. This process is analogous to fluid-only methods that are derived by starting with general continuum stresses together with simplifying assumptions that lead to a pressure equation of state. We will follow a similar strategy, albeit without the fluid-only simplifying assumption by starting with our hyperelastic stress given in (6.5). Although this derivation ultimately yields the commonly used pressure $p = k(\rho - \rho_0)$, where k is a stiffness, ρ is pressure, and ρ_0 is the rest density, that connection must be proven.

6.3.4 Pressure

The problematic term for highly incompressible materials is Ψ_λ. However, we note that this term gives rise to a dilational (constant diagonal) Cauchy stress as

$$\boldsymbol{\sigma}_\lambda = \frac{1}{J}\left(\frac{\partial \Psi_\lambda}{\partial J_E}\frac{\partial J_E}{\partial \boldsymbol{F}_E}\right)\boldsymbol{F}_E^T = \frac{1}{J}\frac{\partial \Psi_\lambda}{\partial J_E} J_E \boldsymbol{F}_E^{-T}\boldsymbol{F}_E^T = -p\mathbf{I}, \tag{6.9}$$

where

$$p = -\frac{1}{J_P}\frac{\partial \Psi_\lambda}{\partial J_E} = -\frac{1}{J_P}\lambda(J_E - 1). \tag{6.10}$$

It is interesting to note that, in the absence of plasticity, $J = J_E$, $J_p = 1$, and $J = \rho/\rho_0$, making (6.10) reduce to $p = -\lambda(\rho/\rho_0 - 1)$, the traditional SPH equation of state [Monaghan, 1992].

6.3.5 Temporal evolution

Even though p is related to our other variables, by treating it as an unknown, we can achieve a splitting, analogous to Chorin-style projection. The main difference is that we are not restricted to fully incompressible materials, and we instead handle the full spectrum. To derive the splitting consider the time evolution of pressure

$$\frac{Dp}{Dt} = -\frac{1}{J_P}\frac{\partial^2 \Psi_\lambda}{\partial J_E^2}\frac{DJ_E}{Dt}. \tag{6.11}$$

Since $J = J_E J_P$ and $\frac{DJ}{Dt} = J\nabla \cdot \boldsymbol{v}$ (e.g., [Gonzalez and Stuart, 2008]), we have $\frac{DJ_E}{Dt} = J_E \nabla \cdot \boldsymbol{v}$ and therefore

$$\frac{Dp}{Dt} = -\frac{1}{J_P}\frac{\partial^2 \Psi_\lambda}{\partial J_E^2} J_E \nabla \cdot \boldsymbol{v} = -\frac{\lambda J_E}{J_P}\nabla \cdot \boldsymbol{v}. \tag{6.12}$$

6.3.6 Discretization

Additionally, with the definition of p from (6.9), our force balance equation takes the fluid-like form

$$\rho\frac{D\boldsymbol{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{g} = \nabla \cdot \boldsymbol{\sigma}_\mu - \nabla p + \rho \boldsymbol{g}, \tag{6.13}$$

where $\boldsymbol{\sigma}_\mu$ is the component of stress from Ψ_μ. We discretize the system of equations (6.12) and (6.13) as

$$\frac{p^{n+1} - p^n}{\Delta t} = -\frac{\lambda^n J_E^n}{J_P^n}\nabla \cdot \boldsymbol{v}^{n+1}, \tag{6.14}$$

$$\frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t} = \frac{1}{\rho^n}\nabla \cdot \boldsymbol{\sigma}_\mu - \frac{1}{\rho^n}\nabla p^{n+1} + \boldsymbol{g}. \tag{6.15}$$

Note that we can replace the material derivative with a simple finite difference in time because advection will be done in a Lagrangian manner using MPM.

To solve the system (6.14) and (6.15), we split the pressure application in (6.15) from the other forces by introducing an intermediate $\boldsymbol{v}^\star$

$$\frac{\boldsymbol{v}^\star - \boldsymbol{v}^n}{\Delta t} = \frac{1}{\rho^n}\nabla \cdot \boldsymbol{\sigma}_\mu + \boldsymbol{g}, \tag{6.16}$$

$$\frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^\star}{\Delta t} = -\frac{1}{\rho^n}\nabla p^{n+1}. \tag{6.17}$$

Taking the divergence of (6.17) and eliminating $\nabla \cdot \boldsymbol{v}^{n+1}$ using (6.14) yields

$$\frac{J_P^n}{\lambda^n J_E^n}\frac{p^{n+1}}{\Delta t} - \Delta t \nabla \cdot \left(\frac{1}{\rho^n}\nabla p^{n+1}\right) = \frac{J_P^n}{\lambda^n J_E^n}\frac{p^n}{\Delta t} - \nabla \cdot \boldsymbol{v}^\star. \tag{6.18}$$

We use $p^n = -\frac{1}{J_P^n}\lambda^n(J_E^n - 1)$ for the right hand side.

Note that the discrete system for the pressure will be symmetric positive definite and similar to a discrete heat equation for moderate λ. As λ is increased to the incompressible limit, the pressure equation is then the standard Poisson equation seen in Chorin-style projections [Chorin, 1968]. This is similar in spirit to the implicit treatment of the compressible Euler equations in [Kwatra et al., 2009]. While the introduction of an auxiliary pressure unknown is common in incompressible elasticity (e.g., [Bonet and Wood, 1997]), it would generally be coupled with the velocity unknowns (e.g., [Mast et al., 2012]). Our introduction of the implicit treatment based on the evolution of pressure (6.11) is novel and drastically improves the efficiency of the approach because it decouples the pressure from the nonlinear equations for velocity unknowns.

6.4 Algorithm

We will now describe the discretization details in our algorithm. We outline each step required to advance one time step in the simulation (Figure 6.1 for a schematic overview). We can think of this process as updating the state (itemized in Table 6.1)

Notation	Description	Is Constant
$\boldsymbol{x}_p$	Position	Not constant
$\boldsymbol{v}_p$	Velocity	Not constant
m_p	Mass	Constant
V_p^0	Initial volume	Constant
$\boldsymbol{F}_{Ep}$	Elastic part of $\boldsymbol{F}_p$	Not constant
$\boldsymbol{F}_{Pp}$	Plastic part of $\boldsymbol{F}_p$	Not constant
μ_p	Lamé parameter μ	Depends on T_p and phase
λ_p	Lamé parameter λ	Depends on T_p, but not phase
T_p	Temperature	Not constant
U_p	Transition heat	Not constant (Sec. 6.4.9)
c_p	Heat capacity per unit mass	Depends on T_p and phase
κ_p	Heat conductivity	Depends on T_p and phase
L_p	Latent heat	Constant
ζ_p	Phase	Depends on T_p and U_p (Sec.6.4.9)

Table 6.1: *Quantities stored on each particle.*

from time t^n to time t^{n+1}. The process uses a background MAC grid and combines standard aspects of traditional MPM and FLIP solvers. Specifically, after the particle state is transferred to the grid, the deviatoric forces are first discretized with implicit MPM in accordance with (6.16). This step results in an intermediate velocity field whose divergence is used in the right hand side of the implicit equation for the dilational part in (6.18). The dilational part is treated with the generalized Chorin-style projection over the MAC grid and the intermediate velocity is then given a pressure correction in accordance with (6.17). The inclusion of the heat transfer effects only requires an additional heat equation solve per time step. We discuss the specific details of each step in the algorithm in the following subsections, which can be summarized as:

1. Apply plasticity from previous timestep (Section 6.4.1)
2. Compute interpolation weights (Section 6.4.2)
3. Rasterize particle data to grid (Section 6.4.3)
4. Classify cells (Section 6.4.4)
5. Compute MPM forces (Section 6.4.5.1)

6. Process grid collisions (Section 6.4.6)
7. Apply implicit MPM update (Section 6.4.5.2)
8. Project velocities (Section 6.4.7)
9. Solve heat equation (Section 6.4.8)
10. Update particle state from grid (Section 6.4.9)
11. Process particle collisions and update particle positions (Section 6.4.10)

6.4.1 Apply plasticity from previous timestep

For simplicity, it is common for graphics researchers to apply a heuristic plastic-yield criterion for compressible elastic materials, because there is considerable leeway in visual applications [Irving et al., 2004; Stomakhin et al., 2013]. However, in the case of nearly incompressible materials, the plastic flow should also be nearly incompressible. We therefore provide a simple procedure for guaranteeing $J_P \equiv \det(\boldsymbol{F}_P) = 1$ for nearly incompressible materials. We note that more accurate plasticity models from the engineering literature (such as von Mises yield criteria) also have the property that $J_P = 1$ as a consequence of rate independence [Bonet and Wood, 1997; Goktekin et al., 2004; Bargteil et al., 2007]. We begin by adjusting $\boldsymbol{F}_E$ and $\boldsymbol{F}_P$ so that the singular values of $\boldsymbol{F}_E$ are restricted to the interval $[1-\theta_c, 1+\theta_s]$ as in [Stomakhin et al., 2013]. We then apply the correction $\boldsymbol{F}_E \leftarrow (J_P)^{1/d}\boldsymbol{F}_E$ and $\boldsymbol{F}_P \leftarrow (J_P)^{-1/d}\boldsymbol{F}_P$, which ensures that $\boldsymbol{F}_P$ is purely deviatoric, or equivalently, $J_P = 1$.

6.4.2 Compute interpolation weights

To transfer data from particles to MAC faces and MAC cell centers, we need multiple sets of interpolation weights per particle. Basically, we have d face-centered grids, one for each dimension, and one cell-centered grid. The procedure of computing the weights is identical for all of these grids and follows [Steffen et al., 2008]. The grids

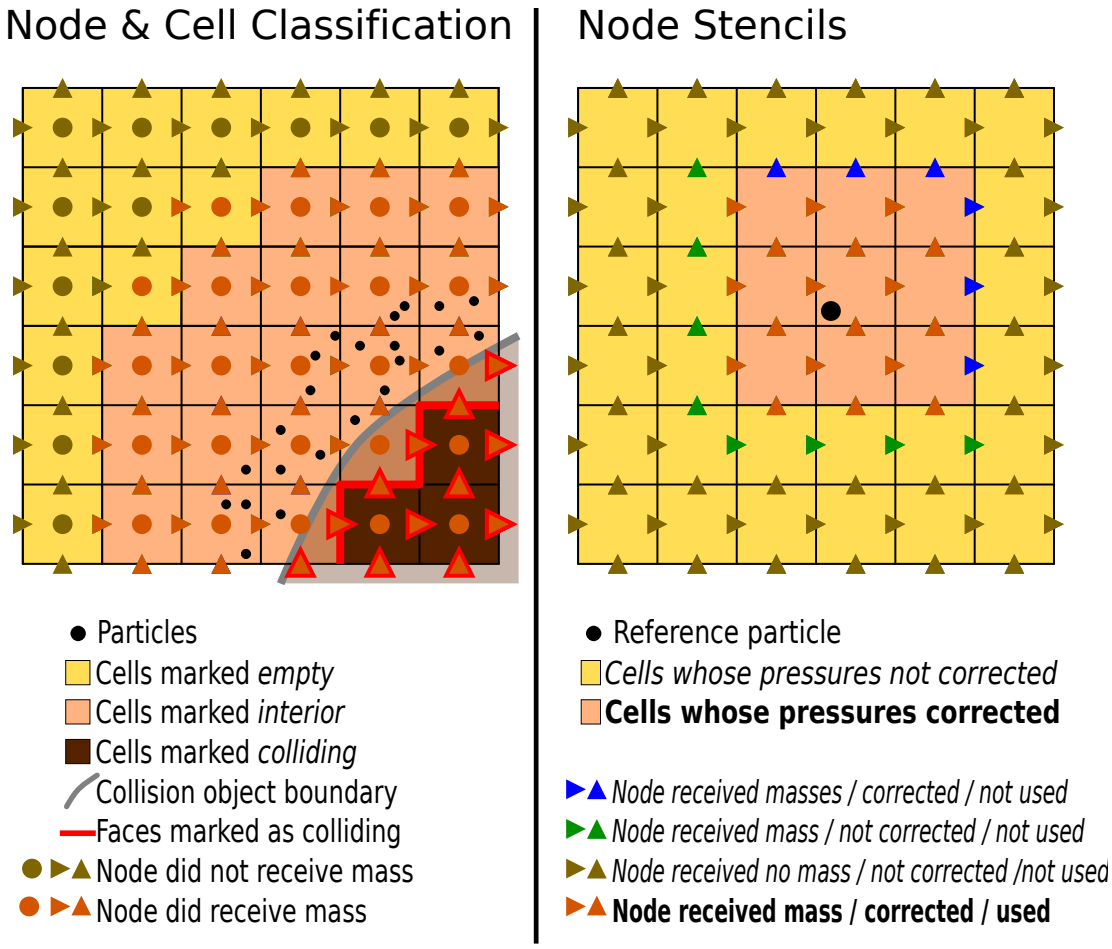

Figure 6.2: *The left figure illustrates cell classification criteria. Note that faces marked "colliding" are Neumann faces for the Poisson solve and yellow cells marked "colliding" are Dirichlet cells for the Poisson solve. The right figure shows stencils for a single reference particle. The particle contributes to the green, blue, and orange faces, the pressure solve only corrects orange and blue faces, but our quadratic interpolation touches only the orange faces.*

however are offset with respect to each other, which leads to different weight values for each grid. Below, we introduce a common notation for all offset grids and describe a way to procedurally calculate the weights.

We express the fact that the $d+1$ grids are offset with respect to each other by considering their base point (x_{0a}, y_{0a}, z_{0a}) (lower-left point in 2D), where $a \in \{x, y, z\}$ indicates velocity components for each of the face grids and $a = \star$ represents the pressure grid. Up to some translation vector, we have $(x_{0x}, y_{0x}, z_{0x}) = (-\frac{h}{2}, 0, 0)$, $(x_{0y}, y_{0y}, z_{0y}) = (0, -\frac{h}{2}, 0)$, $(x_{0z}, y_{0z}, z_{0z}) = (0, 0, -\frac{h}{2})$, and $(x_{0\star}, y_{0\star}, z_{0\star}) = (0, 0, 0)$, where h is the grid spacing. Figure 6.2 (left) illustrates one of the MAC grids. Now, given a grid of spacing h with cell indices $\mathbf{c} = (i, j, k)$ with points located at $\boldsymbol{x}_{\mathbf{c}a} = (x_i, y_j, z_k) = (x_{0a} + ih, y_{0a} + jh, z_{0a} + kh)$ we can define interpolation of an arbitrary particle position $\boldsymbol{x}_p = (x_p, y_p, z_p)$. As in [Steffen et al., 2008], we define a multidimensional separable kernel from the one-dimensional cubic B-spline

$$N(x) = \begin{cases} \frac{1}{2}|x|^3 - x^2 + \frac{2}{3}, & 0 \le |x| < 1 \\ -\frac{1}{6}|x|^3 + x^2 - 2|x| + \frac{4}{3}, & 1 \le |x| < 2 \\ 0, & \text{otherwise} \end{cases} \tag{6.19}$$

as $N^h_{\mathbf{c}a}(\boldsymbol{x}_p) = N(\frac{1}{h}(x_p - x_{ia}))N(\frac{1}{h}(y_p - y_{ja}))N(\frac{1}{h}(z_p - z_{ka}))$.

For a more compact notation, later on we will use $\mathbf{i}$ as an index into MAC grid faces, and $\mathbf{c}$ for indexing cell-centered quantities. E.g., $v_{\mathbf{i}}$ stands for the velocity field component at face $\mathbf{i}$, and $p_{\mathbf{c}}$ is the pressure value at the center of cell $\mathbf{c}$. With this, the interpolation weight of particle $\boldsymbol{x}_p$ is $w_{\mathbf{i}p} = N^h_{\mathbf{c}(\mathbf{i})a(\mathbf{i})}(\boldsymbol{x}_p)$ with respect to face $\mathbf{i}$ and $w_{\mathbf{c}p} = N^h_{\mathbf{c}\star}(\boldsymbol{x}_p)$ with respect to cell $\mathbf{c}$. Here $a(\mathbf{i})$ and $\mathbf{c}(\mathbf{i})$ are the dimension component and cell index associated with face $\mathbf{i}$ respectively. Alternatively, the face index can be uniquely identified by a cell and an axis as $\mathbf{i} = \mathbf{i}(\mathbf{c}, a)$, for $a \in \{x, y, z\}$. Similarly, we define $\nabla w_{\mathbf{i}p} = \nabla N^h_{\mathbf{c}(\mathbf{i})a(\mathbf{i})}(\boldsymbol{x}_p)$ and $\nabla w_{\mathbf{c}p} = \nabla N^h_{\mathbf{c}\star}(\boldsymbol{x}_p)$. The various components and their associated values are summarized in the following table:

Grid	a	Base	Weight
cell	$\star$	$(0,0,0)$	$w_{\mathbf{c}p} = N^h_{\mathbf{c}\star}(\boldsymbol{x}_p)$
x-offset	x	$(-\frac{h}{2},0,0)$	$w_{\mathbf{i}(\mathbf{c},x)p} = N^h_{\mathbf{c}x}(\boldsymbol{x}_p)$
y-offset	y	$(0,-\frac{h}{2},0)$	$w_{\mathbf{i}(\mathbf{c},y)p} = N^h_{\mathbf{c}y}(\boldsymbol{x}_p)$
z-offset	z	$(0,0,-\frac{h}{2})$	$w_{\mathbf{i}(\mathbf{c},z)p} = N^h_{\mathbf{c}z}(\boldsymbol{x}_p)$

6.4.3 Rasterize particle data to grid

We rasterize data to the grid using the interpolation weights from Section 6.4.2. Mass is first rasterized to the grid faces as

$$m^n_{\mathbf{i}} = \sum_p w^n_{\mathbf{i}p} m_p.$$

These face densities allow us to normalize the interpolation of velocity and thermal conductivity as

$$A^n_{\mathbf{i}} = \sum_p w^n_{\mathbf{i}p} m_p A^n_p \quad \text{for} \quad A \in \{v, \kappa\}.$$

We repeat the process at cell centers, computing cell masses $m^n_{\mathbf{c}} = \sum_p w^n_{\mathbf{c}p} m_p$ followed by

$$B^n_{\mathbf{c}} = \frac{1}{m^n_{\mathbf{c}}} \sum_p w^n_{\mathbf{c}p} m_p A^n_p \quad \text{for} \quad B \in \{J, J_E, c, T, \lambda^{-1}\},$$

noting that rasterizing λ^{-1} rather than λ is important for stability.[1] Finally, we set $J^n_{P\mathbf{c}} = \frac{J^n_{\mathbf{c}}}{J^n_{E\mathbf{c}}}$.

[1] The relationship between J_E and λ results in a balance in the pressure $-\frac{\lambda}{J_P}(J_E - 1)$. Unfortunately, averaging J_E and λ through rasterization might destroy this balance, creating an artificially large pressure. Estimating λ with a harmonic average, or equivalently, rasterizing λ^{-1} and computing $\lambda_{\mathbf{c}} = 1/\lambda^{-1}_{\mathbf{c}}$, resolves this problem.

6.4.4 Classify cells

We represent our collision objects as level sets and assign each collision object a temperature. We begin the collision processing by checking all faces for collisions. A MAC face is colliding if the level set computed by any collision object is negative at the face center. If it is colliding, we flag the face as colliding. For convenience and consistency in other parts of the algorithm, we classify each MAC cell as *empty*, *colliding*, or *interior*. A cell is marked as colliding if all of its surrounding faces are colliding. Otherwise, a cell is interior if the cell and all of its surrounding faces have mass. All remaining cells are empty (left portion of Figure 6.2). Colliding cells are either assigned the temperature of the object it collides with or a user-defined spatially-varying value, depending on the setup. If the free surface is being enforced as a Dirichlet temperature condition, the ambient air temperature is recorded for empty cells. No other cells require temperatures to be recorded at this stage.

6.4.5 MPM velocity update

In our deviatoric/dilational splitting of the material response, the deviatoric forces are discretized with implicit MPM, and the dilation part is discretized with the generalized Chorin-style projection. Using the common notation from a projection method, we can think of the the implicit MPM step as updating rasterized grid-based velocities $v_{\mathbf{i}}^n$ to $v_{\mathbf{i}}^\star$ in accordance with (6.16). The last step for grid velocities is to apply the pressure correction, computed using (6.18), to $v_{\mathbf{i}}^\star$ to obtain $v_{\mathbf{i}}^{n+1}$ in accordance with (6.17). We outline the procedure for computing $v_{\mathbf{i}}^\star$ in this section and the following subsections. The first step is to compute the MPM force.

Following Stomakhin et al. [2013], we discretize the deviatoric forces via a potential energy. This naturally facilitates an implicit treatment with symmetric linearization. We denote the location of grid face $\mathbf{i}$ as $\boldsymbol{x}_{\mathbf{i}}$. If we interpret our Eulerian MAC grid as though

it were Lagrangian, we would estimate that after Δt, this face would have moved to $\hat{\boldsymbol{x}}_{\mathbf{i}} = \boldsymbol{x}_{\mathbf{i}} + \Delta t v^{\star}_{\mathbf{i}} \mathbf{e}_{a(\mathbf{i})}$, where $\mathbf{e}_{a(\mathbf{i})}$ is the basis vector in the direction corresponding to the MAC velocity component $v^{\star}_{\mathbf{i}}$. If we denote the vector of all $\hat{\boldsymbol{x}}_{\mathbf{i}}$ as $\hat{\boldsymbol{x}}$, then we can think of it as depending on the vector of all face velocities $v^{\star}_{\mathbf{i}}$ which we can denote as $\boldsymbol{v}^{\star}$. Or, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(\boldsymbol{v}^{\star})$. Note that this interpretation is for convenience in computing forces and force derivatives as we do not actually move our grid. Since we only really have one degree of freedom in $\hat{\boldsymbol{x}}_{\mathbf{i}}$, we will denote it as $\hat{x}_{\mathbf{i}} = (\hat{\boldsymbol{x}}_{\mathbf{i}})_{a(\mathbf{i})}$ and $\hat{x}_{\mathbf{i}} = \hat{x}_{\mathbf{i}}(v^{\star}_{\mathbf{i}}) = (\hat{\boldsymbol{x}}_{\mathbf{i}})_{a(\mathbf{i})} + \Delta t v^{\star}_{\mathbf{i}}$.

The deviatoric potential energy is

$$\Phi_{\mu}(\hat{\boldsymbol{x}}) = \sum_{p} V_p^0 \hat{\Psi}_{\mu}(\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}})), \tag{6.20}$$

where V_p^0 is the initial volume occupied by particle p and $\hat{\boldsymbol{F}}_{Ep}$ is the elastic part of the deformation gradient of particle p. $\hat{\boldsymbol{F}}_{Ep}$ depends on $\hat{\boldsymbol{x}}$ as in [Sulsky et al., 1995]

$$\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}) = \left(\mathbf{I} + \sum_{\mathbf{i}} (\hat{\boldsymbol{x}}_{\mathbf{i}} - \boldsymbol{x}_{\mathbf{i}})(\nabla w^n_{\mathbf{i}p})^T \right) \boldsymbol{F}^n_{Ep}. \tag{6.21}$$

6.4.5.1 MPM forces

The force component $f_{\mathbf{i}}$ at face $\mathbf{i}$ is given by $f_{\mathbf{i}} = -\frac{\partial \Phi}{\partial \hat{x}_{\mathbf{i}}} = -\frac{\partial \Phi}{\partial \hat{\boldsymbol{F}}_{Ep}} \frac{\partial \hat{\boldsymbol{F}}_{Ep}}{\partial \hat{\boldsymbol{x}}_{\mathbf{i}}} \frac{\partial \boldsymbol{x}_{\mathbf{i}}}{\partial x_{\mathbf{i}}}$, or

$$f_{\mathbf{i}}(\hat{\boldsymbol{x}}) = -\sum_{p} V_p^0 \mathbf{e}^T_{a(\mathbf{i})} \frac{\partial \Psi}{\partial \boldsymbol{F}_E}(\hat{\boldsymbol{F}}_{Ep}(\hat{\boldsymbol{x}}))(\boldsymbol{F}^n_{Ep})^T \nabla w^n_{\mathbf{i}p}. \tag{6.22}$$

With these forces, we compute the right hand side for our MPM treatment

$$b_{\mathbf{i}} = v^n_{\mathbf{i}} + \frac{\Delta t}{m_{\mathbf{i}}} f_{\mathbf{i}} + \Delta t g_{\mathbf{i}} \sum_{p} w^n_{\mathbf{i}p}, \tag{6.23}$$

where $g_{\mathbf{i}}$ is the gravity component at face $\mathbf{i}$ and $f_{\mathbf{i}}^n = -\frac{\partial \Phi_\mu}{\partial \hat{x}_{\mathbf{i}}}(\hat{\boldsymbol{x}}(\mathbf{0}))$, again using the convention that $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(\boldsymbol{v}^\star)$.

6.4.5.2 Semi-implicit MPM update

We use one step of Newton's method to solve the implicit system for deviatoric and inertial force balance. This yields a (mass) symmetric system for $\boldsymbol{v}^\star$

$$\sum_{\mathbf{j}} \underbrace{\left(\delta_{\mathbf{ij}} + \frac{\Delta t^2}{2m_{\mathbf{i}}^n} \frac{\partial^2 \Phi_\mu}{\partial \hat{x}_{\mathbf{i}} \partial \hat{x}_{\mathbf{j}}}(\hat{\boldsymbol{x}}(\mathbf{0}))\right)}_{q_{\mathbf{ij}}} v_{\mathbf{j}}^\star = b_{\mathbf{i}}, \tag{6.24}$$

where $q_{\mathbf{ij}}$ are the entries of matrix $\boldsymbol{Q}$. The system is symmetric but potentially indefinite, so we use the iterative conjugate residual method [Choi, 2006]. This Krylov method only requires the action of $\boldsymbol{Q}$ on an arbitrary increment $\delta\mathbf{u}$ (comprised of scalar MAC face increments $\delta u_{\mathbf{j}}$). The nontrivial term is from the Hessian and can be expressed as

$$-\delta f_{\mathbf{i}} = \sum_{\mathbf{j}} \frac{\partial^2 \Phi_\mu}{\partial \hat{x}_{\mathbf{i}} \partial \hat{x}_{\mathbf{j}}}(\hat{\boldsymbol{x}}(\mathbf{0}))\delta u_{\mathbf{j}} = \sum_p V_p^0 \mathbf{e}_{a(\mathbf{i})}^T \mathbf{A}_p (\boldsymbol{F}_{Ep}^n)^T \nabla w_{\mathbf{i}p}^n, \tag{6.25}$$

where

$$\mathbf{A}_p = \frac{\partial^2 \Psi_\mu}{\partial \boldsymbol{F}_E^2}(\boldsymbol{F}_E(\hat{\boldsymbol{x}})) : \left(\sum_{\mathbf{j}} \delta u_{\mathbf{j}} \mathbf{e}_{a(\mathbf{i})} (\nabla w_{\mathbf{j}p}^n)^T \boldsymbol{F}_{Ep}^n\right). \tag{6.26}$$

6.4.6 Process the grid collisions

Each face marked as colliding during the cell classification step must have its velocity corrected for collisions. We perform sticking collisions for all of our collisions, so we simply assign the velocity component from the collision body to the corresponding face on the MAC grid.

6.4.7 Project the velocities

We discretize (6.18) for the pressure then use it to correct the intermediate velocities $\boldsymbol{v}^\star$. This is a discrete parabolic equation that, of course, reduces to a Poisson equation in the incompressible limit of $\lambda \to \infty$. In either case our discretization reduces to a symmetric positive definite system of equations. We discretize in space using the central-difference stencils naturally defined over the MAC grid. The right hand side of our system has entries $s_{\mathbf{c}}$ stored at MAC cell centers. We compute these as $s_{\mathbf{c}} = -\frac{J_{E\mathbf{c}}^n - 1}{\Delta t J_{E\mathbf{c}}^n} - \sum_{\mathbf{i}}^n G_{\mathbf{ic}} v_{\mathbf{i}}^\star$, where $G_{\mathbf{ic}}$ are the coefficients of the central-differenced gradient stencil. Our corresponding matrix takes the increments $\delta p_{\mathbf{c}}$ and produces the results $\delta r_{\mathbf{c}}$, where $\delta r_{\mathbf{c}} = \frac{\delta p_{\mathbf{c}} J_{P\mathbf{c}}^n}{J_{E\mathbf{c}}^n \lambda_{\mathbf{c}}^n \Delta t} + \Delta t \sum_{\mathbf{i}} \sum_{\mathbf{c}'} \frac{1}{\rho_{\mathbf{i}}^n} G_{\mathbf{ic}} G_{\mathbf{ic}'} \delta p_{\mathbf{c}'}$ and $\rho_{\mathbf{i}}^n = \frac{m_{\mathbf{i}}^n}{V_{\mathbf{i}}^n}$ is the mass density at face $\mathbf{i}$. The mass at the face is $m_{\mathbf{i}}^n$, and $V_{\mathbf{i}}^n$ is a control volume around the face, whose formula we describe below. Once we have solved for the pressure, we apply the pressure correction to the velocities using $v_{\mathbf{i}}^{n+1} = v_{\mathbf{i}}^\star - \Delta t \sum_{\mathbf{c}} \frac{1}{\rho_{\mathbf{i}}^n} G_{\mathbf{ic}} p_{\mathbf{c}}$.

The discretization of $G_{\mathbf{ic}}$ corresponds to a simple voxelized, central-differenced gradient operator. We enforce homogeneous Dirichlet pressure boundary conditions at cells that have been marked as empty, and homogeneous Neumann boundary conditions at faces adjacent to cells that have been marked as colliding.

Degrees of freedom near collision objects do not have as many neighboring particles as interior degrees of freedom, since part of their influence is covered by a collision object. This causes these faces to appear lighter, which would in turn cause them to rise under gravity without careful definition of $\rho_{\mathbf{i}}^n$. We prevent such phenomena by computing control volumes that accurately represent the portion of the domain associated with a face. This is done as $V_{\mathbf{i}}^n = \sum_{\mathbf{c}} \int_{\Omega_{\mathbf{c}}} \chi_{\mathbf{c}} N_{\mathbf{c}(\mathbf{i})a(\mathbf{i})}^h(\boldsymbol{x}) d\boldsymbol{x}$, where $\Omega_{\mathbf{c}}$ is the interior of MAC cell $\mathbf{c}$, and $\chi_{\mathbf{c}} = 1$ if cell $\mathbf{c}$ is marked as interior or $\chi_{\mathbf{c}} = 0$ otherwise. This is an approximation to $\int_{\Omega^n} N_{\mathbf{i}}^h d\boldsymbol{x}$, where Ω^n is the domain encompassed by the material. This control volume is essential for accurately approximating the density near collision

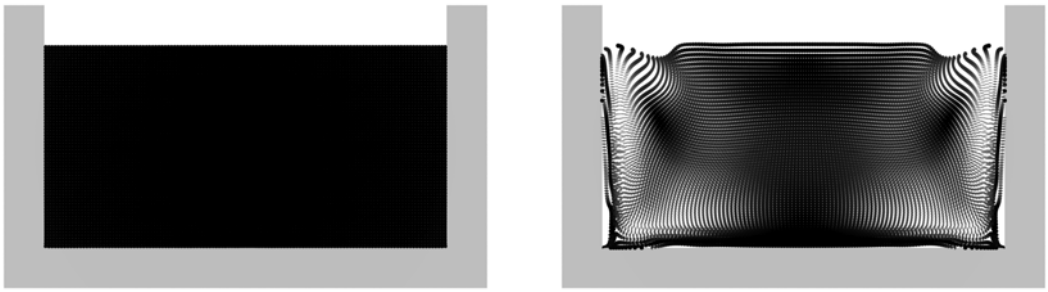

Figure 6.3: *Simulation of a stationary pool with (left) and without (right) density correction. Without correction the faces near collision objects appear lighter which causes them to rise under gravity.*

objects. Note that the integral described in the formula for $V_{\mathbf{i}}^n$ has only a finite number of cases, which the product structure of $N^h_{\mathbf{c(i)}a(\mathbf{i})}(\boldsymbol{x})$ makes relatively easy to tabulate. We demonstrate the effect of density correction in Figure 6.3.

6.4.8 Solve the heat equation

We perform a stabilized Poisson solve to update the temperature in accordance with (6.4). We begin by setting the right hand side to $T_{\mathbf{c}}^n$, which is a cell-centered rasterized temperature. Our corresponding matrix takes the increments $\delta T_{\mathbf{c}}$ and produces the result $\delta T_{\mathbf{c}} + \Delta t \sum_{\mathbf{i}}^n \sum_{\mathbf{c}'} \frac{\Delta x^d}{m_{\mathbf{c}}^n c_{\mathbf{c}}^n} \kappa_{\mathbf{i}}^n G_{\mathbf{ic}} G_{\mathbf{ic}'} \delta T_{\mathbf{c}'}$. The discretization of $G_{\mathbf{ic}}$ corresponds to a simple voxelized, central-differenced gradient operator. We enforce Dirichlet temperature boundary conditions at cells that are in contact with fixed temperature bodies (like a heated pan or air) and homogeneous Neumann boundary conditions at faces adjacent to cells that can be considered empty or corresponding to insulated objects.

6.4.9 Update the particle state from the grid

Some outermost faces involved in the MPM step do not receive a correction from the projection step, and as a result they tend to have outdated velocity values (Figure 6.2).

To prevent errors from uncorrected velocities when transferring information back to the particles, we use a tighter quadratic stencil given by the following spline:

$$N(x) = \begin{cases} -x^2 + \frac{3}{4}, & 0 \leq |x| < \frac{1}{2} \\ \frac{1}{2}x^2 - \frac{3}{2}x + \frac{9}{8}, & 1 \leq |x| < \frac{3}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.27}$$

We interpolate velocities back to particles using FLIP, where the PIC component is computed as $\boldsymbol{v}_p^{PIC} = \sum_{\mathbf{i}} v_{\mathbf{i}}^{n+1} w_{\mathbf{i}p}^n \mathbf{e}_{a(\mathbf{i})}$ and the FLIP component as $\boldsymbol{v}_p^{FLIP} = \boldsymbol{v}_p^n + \sum_{\mathbf{i}} (v_{\mathbf{i}}^{n+1} - v_{\mathbf{i}}^n) w_{\mathbf{i}p}^n \mathbf{e}_{a(\mathbf{i})}$. With these, the new velocities are $\boldsymbol{v}_p^{n+1} = \alpha \boldsymbol{v}_p^{FLIP} + (1 - \alpha)\boldsymbol{v}_p^{PIC}$, where α is the FLIP fraction. We used $\alpha = 0.95$ in our examples.

The next step is to update $\boldsymbol{F}_{Ep}$. To do this, we must compute a velocity gradient, which we do with $\nabla \boldsymbol{v}_p^{n+1} = \sum_{\mathbf{i}} v_{\mathbf{i}}^{n+1} \mathbf{e}_{a(\mathbf{i})} \nabla w_{\mathbf{i}p}^T$. Normally, one would finish with the update rule $\boldsymbol{F}_{Ep}^{n+1} = (\mathbf{I} + \Delta t \nabla \boldsymbol{v}_p^{n+1}) \boldsymbol{F}_{Ep}^n$. We found that this occasionally leads to $J_{Ep}^{n+1} \leq 0$ if the time step is too large, so we opt for a compromise between this simple rule and the ideal but expensive exponential computation $\boldsymbol{F}_{Ep}^{n+1} = e^{\Delta t \nabla \boldsymbol{v}_p^{n+1}} \boldsymbol{F}_{Ep}^n$. Instead, we use $\boldsymbol{F}_{Ep}^{n+1} = \boldsymbol{R}(\Delta t \nabla \boldsymbol{v}_p^{n+1}) \boldsymbol{F}_{Ep}^n$, where $\boldsymbol{R}(\boldsymbol{M}) = \mathbf{I} + \boldsymbol{M}$ if $\det(\mathbf{I} + \boldsymbol{M}) > 0$ and $\boldsymbol{R}(\boldsymbol{M}) = \boldsymbol{R}(\frac{1}{2}\boldsymbol{M})^2$ otherwise. Note that this is effectively a truncated geometric series of the exponential function, where we invest just enough time to keep the determinant positive. In practice, this function recurses very rarely, and the update is more robust but nearly as efficient as before. If p is a fluid particle, we finish off the update of $\boldsymbol{F}_{Ep}^{n+1}$ by removing its deviatoric component using $\boldsymbol{F}_{Ep}^{n+1} \leftarrow (J_{Ep}^{n+1})^{1/d} \mathbf{I}$.

Similarly to velocity, temperature gets transferred from the grid cell centers to particles as $T_p^{n+1} = \beta T_P^{FLIP} + (1 - \beta) T_P^{PIC}$, where $T_p^{FLIP} = T_p^n + \sum_{\mathbf{c}} (T_{\mathbf{c}}^{n+1} - T_{\mathbf{c}}^n) w_{\mathbf{c}p}$, $T_p^{PIC} = \sum_{\mathbf{c}} T_{\mathbf{c}}^n w_{\mathbf{c}p}$ and β is the FLIP ratio (we used $\beta = 0.95$ for our examples). As mentioned before, the heat equation and, thus, the grid-based heat update are valid only within one material phase, so cases where the temperature crosses the freezing

point require special treatment. Namely, a portion of the heat the particle gets (or loses) should be spent on the phase change. To account for this effect, we have an energy buffer associated with each particle of size L_p, and the particle stores the amount of heat U_p contained in that buffer, which can vary from 0 to L_p. Initially, we allow each particle to freely change its temperature according to the heat equation. But whenever the freezing point is reached, any additional temperature change is multiplied by $c_p m_p$ and added to the buffer, with the particle temperature kept unchanged. This can also be viewed as a post correction of the temperature for a particle that "illegally" crossed the freezing point. Once the buffer is completely full (particle heat $U_p = L_p$), we switch the particle phase to fluid. Conversely, if the buffer becomes empty (particle heat $U_p = 0$), we switch the particle phase to solid. Note that the phase change happens *only* when the buffer is completely full or empty, otherwise the material retains its phase from the previous timestep. This sort of hysteresis facilitates more stable phase transitions, as opposed to using a hard threshold on U_p.

6.4.10 Process the particle collisions and positions

We complete our time integration by enforcing collisions on our particles. Since we did sticking collisions with the grid, we do sticking collisions on particles as well. A particle is registered as colliding if a collision body registers a negative level set value at the location of the particle. If this occurs, the particle's velocity is set to the velocity of the collision body at that location. Finally, we update particle positions as $\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \boldsymbol{v}_p^{n+1}$.

6.5 Simulation results

We have generated a number of visually interesting results using our method. Our novel splitting and rasterization techniques facilitate handling mixtures with extreme varia-

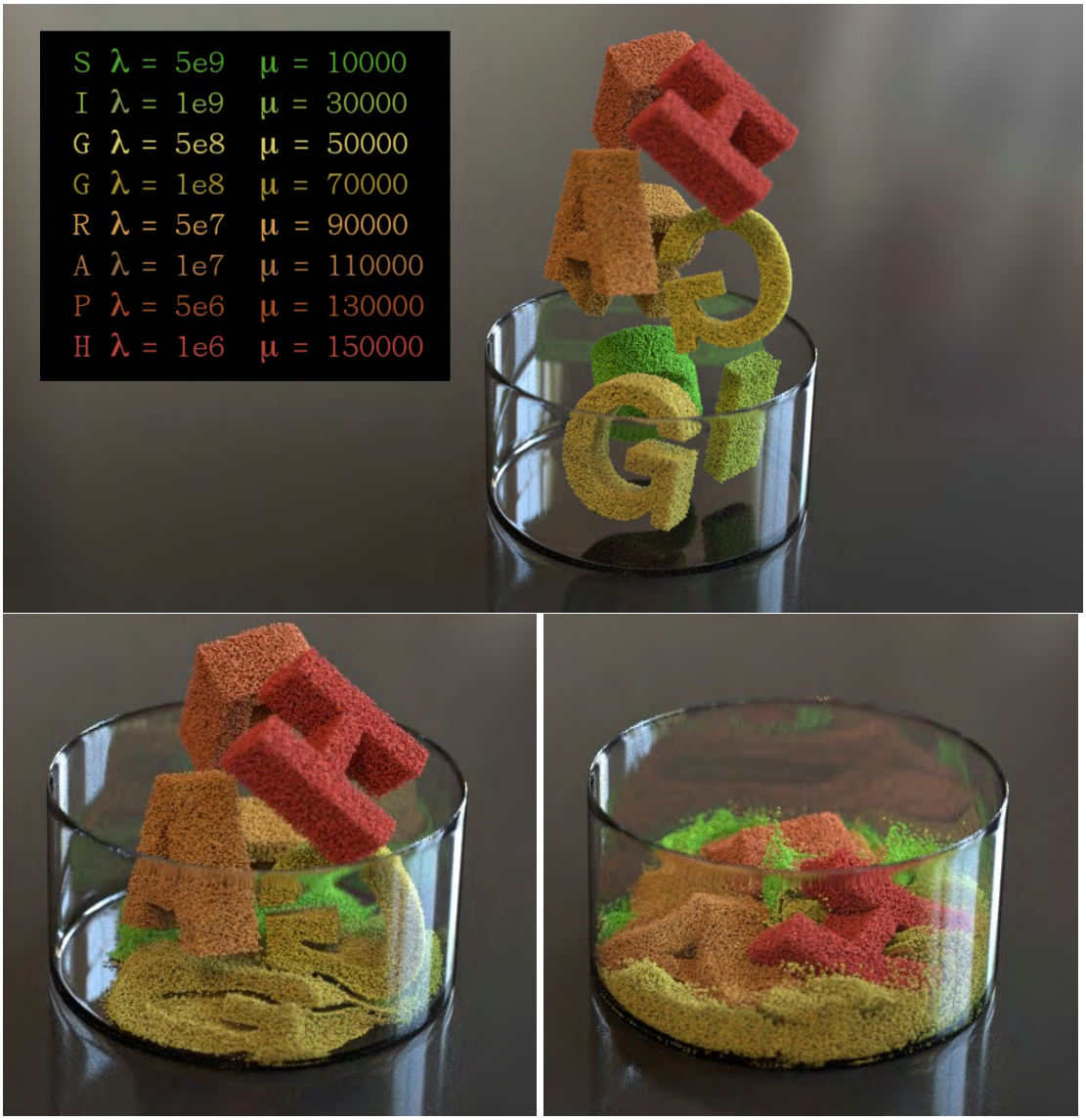

Figure 6.4: *Our method handles mixtures of materials with drastically different properties, ranging from compressible to (almost) incompressible. Here each letter has λ varying from 10^6 to 5×10^9, as well as varying μ and plasticity parameters.* ©*Disney.*

Figure 6.5: *Bringing a hot fluid stream in contact with a cold solid produces compelling phase transition effects. The image demonstrates both: Simulated particle view with temperature distribution (top) and the rendering of our final meshed geometry (bottom).* ©*Disney.*

tions of material parameters. This can be seen in Figure 6.4 where we drop elastoplastic SIGGRAPH letters with material properties ranging from compressible to almost incompressible with varying stiffness and plasticity parameters.

Further, our simplified yet practical heat model allows us to achieve compelling phase transition effects. Figure 6.5 shows hot liquid chocolate pouring on a cold solid chocolate bunny. During the process some solid melts and some liquid freezes producing intricate shapes. Figure 6.6 and Figure 6.7 demonstrate how we can use external surface heat sources and sinks (such as hot/cold air and the cold frying pan) to melt and freeze different objects. Our careful treatment of the physics of phase transition using latent heat allows us to maintain sharp, yet stable, interfaces between solid and fluid phases, as can be seen in the butter melting example in Figure 6.8. By varying materials' thermal parameters such as heat conductivity, heat capacity, and latent heat, we can control the heat flow and thus (indirectly) affect the dynamics of melting and freezing, as shown in Figure 6.9. To create believable lava flow solidifying into pāhoehoe shown in Figure 6.10, we varied the temperature of the mountain based on the distance to the lava source (the heat exchange with the air was not simulated). This way the lava would freeze more gradually, forming attractive layered shapes. We also added some variation to the freezing temperature of the particles in order to give it a more amorphous look.

Figure 6.6: *Setting a Dirichlet temperature boundary condition on the air cells allows us to melt objects from the outside.* ©*Disney.*

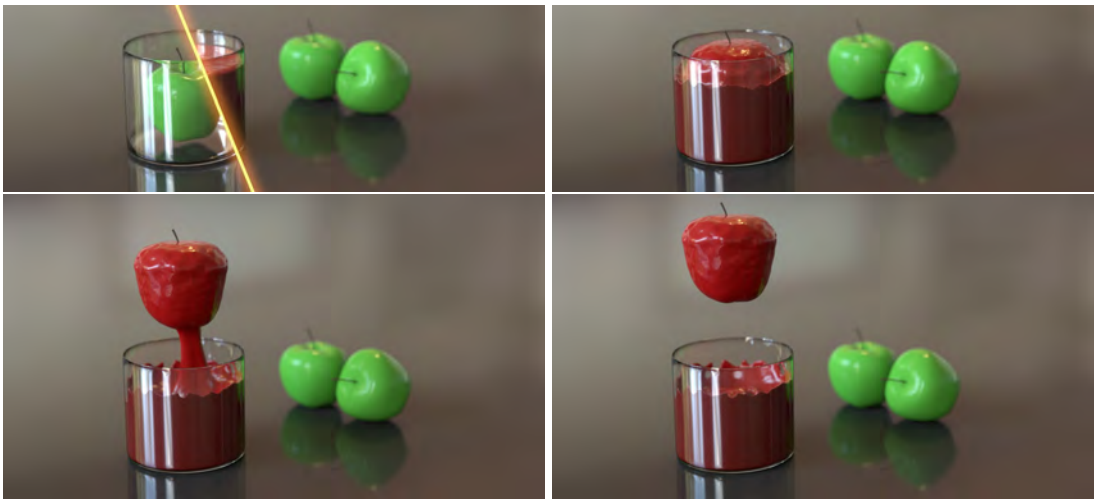

Figure 6.7: *An apple is pulled from liquid candy and it hardens on contact with the air, creating a sticky, candied apple.* ©*Disney.*

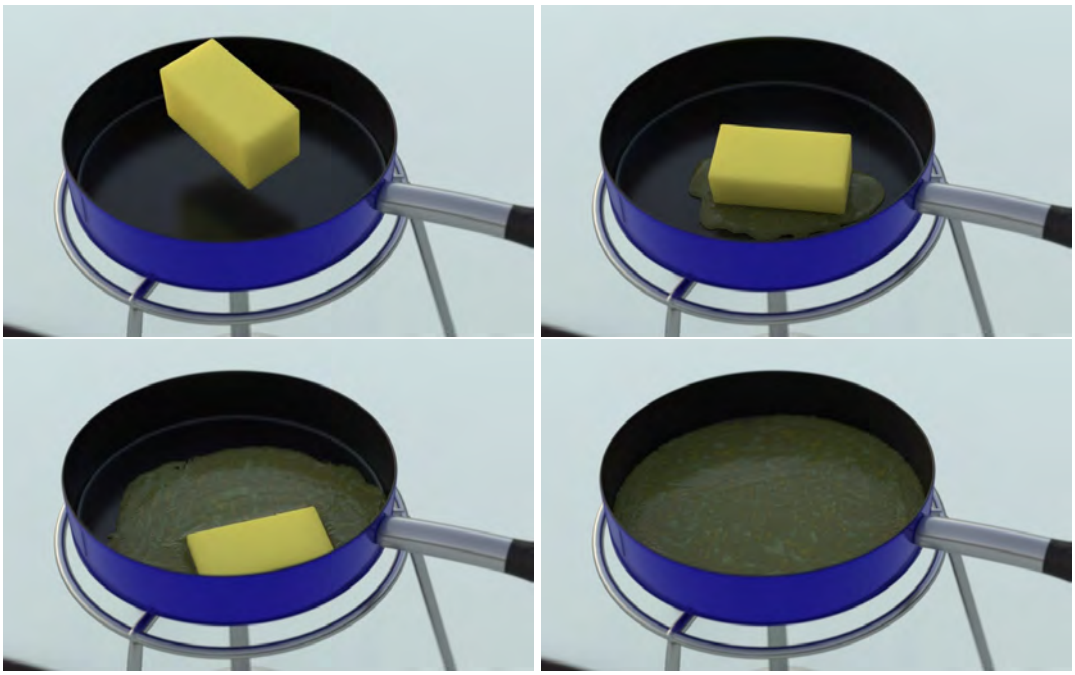

Figure 6.8: *Our method is able to capture many intricate features of butter melting over a hot frying pan, such as wave-like spread and micro ripples of the fluid phase, as well as effortless sliding behavior of the solid chunk on top of it.* ©*Disney.*

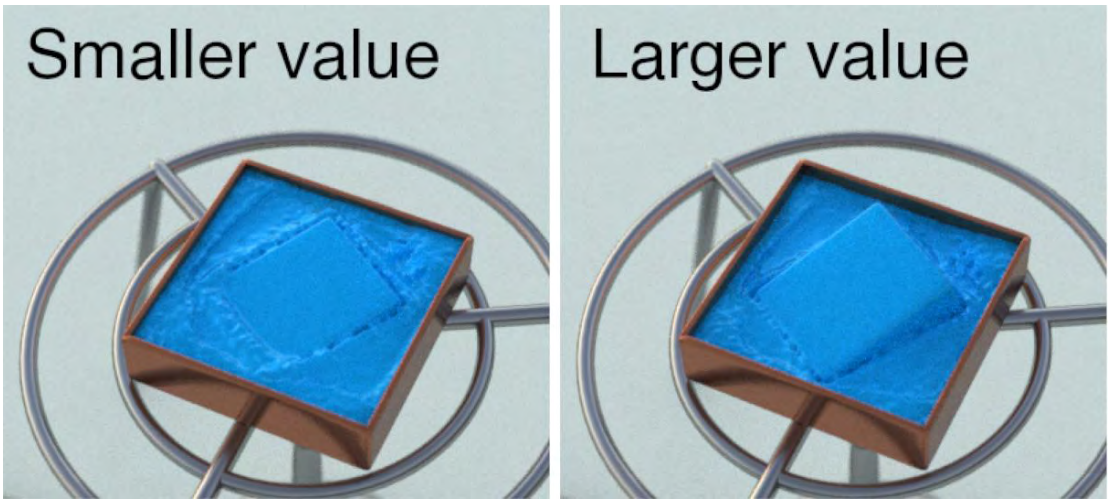

Figure 6.9: *Changing the value of latent heat affects the rate of phase transition, demonstrated by this melting wax example.* ©*Disney.*

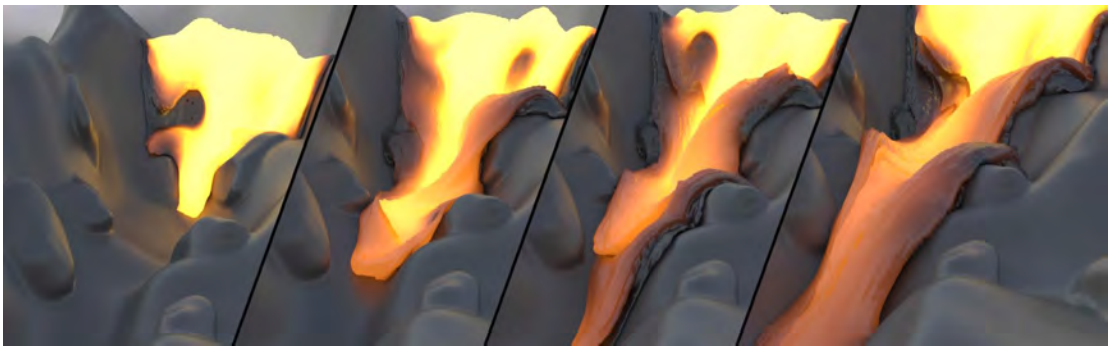

Figure 6.10: *Lava solidifying into pāhoehoe forms complex and attractive shapes. The lava emits light according to the blackbody spectrum corresponding to the simulated temperature.* ©*Disney.*

Example	Particles	Grid	min/frame
SIGGRAPH letters	1.0×10^6	$96 \times 144 \times 96$	18.5
Bunny and hot stream	1.2×10^6	$170 \times 170 \times 170$	8.4
Bunny and hot air	1.2×10^6	$160 \times 160 \times 160$	11.4
Apple dip	0.8×10^6	$64 \times 128 \times 64$	11.0
Melting butter	4.2×10^6	$128 \times 128 \times 128$	14.5
Lava	3.5×10^6	$300 \times 150 \times 300$	29.7

Table 6.2: *Particle counts, grid resolutions and simulation times per frame for each of our examples. Simulations were performed on a 16-core Xeon E5-268 2.67 GHz machine.*

The simulation times for each of the examples are shown in Table 6.2. For each of those, the timestep size was $\Delta t \simeq 3 \times 10^{-4}$ s. To achieve convergence, the conjugate residual solvers for the MPM, projection, and heat diffusion steps normally would take under 10, 300 and 50 iterations, respectively.

6.6 Discussion

MPM: While MPM yields automatic collision and topology changes, it incurs some difficulties. For example, the grid introduces numerical plasticity, and it is difficult to represent sharp interfaces between materials. One down-side of our cubic interpolant is that we have a wider stencil compared to what most incompressible FLIP solvers use. This leads to additional numerical viscosity as well as increased computational expense. While it is tempting to use quadratic B-splines for rasterization paired with trilinear interpolation, low-order interpolation with the MPM is known to have stability problems [Steffen et al., 2008]. Additionally, we focused on sticky boundaries because the materials we were simulating were typically sticky. Thus, deriving a free-slip boundary condition would be interesting future work. It would be interesting to consider alternative integration strategies that would yield bigger time steps, though our time steps tend to be commensurate with [Stomakhin et al., 2013].

Projection: Although the projection-like decoupling of pressure from MPM-discretized deviatoric terms is valid away from the boundary, there is still coupling through the free surface boundary condition $\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{\sigma}_\mu \cdot \boldsymbol{n} - p\boldsymbol{n} = \mathbf{0}$. In order to separate the MPM-based solution of the deviatoric terms from the pressure equations, we implicitly assume that $\boldsymbol{\sigma}_\mu \cdot \boldsymbol{n} = \mathbf{0}$ during the MPM solve and then that $p = 0$ for the projection step, at the free surface. While this does guarantee that $\boldsymbol{\sigma} \cdot \boldsymbol{n} = \boldsymbol{\sigma}_\mu \cdot \boldsymbol{n} - p\boldsymbol{n} = \mathbf{0}$, it removes some flexibility as it is akin to enforcing $a + b = 0$ with $a = 0$ and $b = 0$. Note that the boundary condition $\boldsymbol{\sigma}_\mu \cdot \boldsymbol{n} = \mathbf{0}$ is automatically enforced at the free surface with

an MPM discretization since it is the “natural" boundary from the variational principle on which MPM is based. While this decoupling certainly causes errors in both pressure and the velocities (see, e.g., [Hirt and Shannon, 1968] for discussion), this simplification is commonly done in both the computer graphics [Carlson et al., 2002b; Goktekin et al., 2004; Rasmussen et al., 2004; Losasso et al., 2006b; Batty and Bridson, 2008b] and engineering literature [Harlow and Welch, 1965].

Performance: Our implementation was parallelized and has shown good scaling results with increasing number of CPU cores. However, the performance still remains an issue. In particular, the grid rasterization step (including matrix-vector multiplication in the implicit MPM solve) constitutes a significant portion of runtime. In the future, we might consider acceleration via CPU SIMD or GPGPU techniques to improve the performance. Also, employing simulation level of detail techniques could reduce run times in areas where the particles have settled. Alternatively, Lagrangian techniques such as [Solenthaler et al., 2007] have achieved material variation and melting effects with less computational cost. Nevertheless, we believe our formulation remains interesting because it provides a theoretical unification between two popular algorithms while also allowing formalized constitutive modeling.

Sampling: Particle methods can suffer from poor sample quality under large deformation. Even though pure Lagrangian methods can avoid drift when returning to a rest configuration, under significant plastic deformation, conditioning, sample density, and accuracy may degrade, requiring remeshing (e.g., [Bargteil et al., 2007]) or resampling. While we note that in the presence of less liquid-like behavior, drift is less of an issue. We plan to experiment with resampling techniques in the future.

Rendering: While modeling and simulation is simplified with particle methods, obtaining high-quality rendering becomes more challenging. Since the MPM naturally produces a density rasterization, index-of-refraction matched volume renderers can sometimes be applied (e.g., for snow). For most of the materials, however, we needed

to render an interface, thus we turned to meshing solutions. Such techniques are common for liquid rendering and typically involve rasterizing particles to a grid using some (usually spherically symmetric) basis function followed by grid smoothing, contouring, and final surface smoothing. These steps typically require per-example tuning and it is often impossible to recover as much detail as the particles seem to possess. This can be seen in Figure 6.5. We also experimented with anisotropic kernel techniques such as [Yu and Turk, 2010], but we found that while they are very successful for liquids with visible surface tension, in our case they created more artifacts than they removed. Thus, any techniques that improve meshing will improve the final quality of our results.

6.7 Summary

In summary, we introduced a novel material point method for melting and solidifying materials using a heat solver to capture the underlying thermodynamics and alter mechanical parameters. The method is implicit and capable of simulating nearly incompressible materials using a Chorin-like projection solve. Hence, we have widened the range of materials that the MPM can handle, and we have demonstrated this breadth with several compelling melting and solidifying examples.

CHAPTER 7

Conclusion

The multiphysics simulation of materials undergoing large deformation and topology change is useful in Computer Graphics, Mechanical Engineering, and other fields. In these situations, how to cleverly represent material geometry as well as accurately resolve its evolution under energetic external loads or subject to heat transport remains an important and challenging problem. This thesis focused on the development of novel hybrid, Lagrangian/Eulerian simulation methods. In these methods, particles are used to resolve transport and topological change while a background Eulerian grid is used to compute mechanical forces and collision responses. Particle-in-Cell (PIC) techniques, particularly the Fluid Implicit Particle (FLIP) variants, including the Material Point Method (MPM), have become the norm in computer graphics fluid simulation. Our approach effectively reduces numerical dissipation and overcomes instabilities in solving energetic mechanical problems. We developed an augmented MPM for simulating multiphase materials integrated with heat transportation. This method is capable of handling the underlying thermodynamics in coupled simulations of phase-changing materials. Each method in this dissertation independently solves an existing problem or proposes a novel approach to certain simulation scenarios. Together, they contribute to the unified goal of simulating dynamic-topology materials as well as the coupling of them.

We called our new hybrid, Lagrangian/Eulerian method the Affine Particle-In-Cell method (APIC). While existing approaches (PIC, FLIP, MPM) have proven very pow-

erful, they do suffer from some well-known limitations. The original PIC is stable, but highly dissipative, while FLIP, which is designed to remove this dissipation, is more noisy and at times unstable. We presented a novel technique designed to retain the stability of the original PIC, without suffering from the noise and instability of FLIP. Our key observation was that the dissipation in the original PIC results from a loss of information when transferring between grid and particle representations. We prevented this loss of information by augmenting each particle with a locally affine, rather than locally constant, description of the velocity. We showed that this not only stably removes the dissipation of PIC, but that it also enables exact conservation of angular momentum across the transfers between the particles and the grid. With our method, we controlled noise by keeping the pure filter property of PIC but minimized information loss by enriching each particle with a tiny matrix providing a locally affine (rather than a locally constant) description of the flow. Our APIC method effectively reduces dissipation, preserves angular momentum and prevents instabilities. Furthermore, we demonstrated that the method is applicable to both incompressible liquids and MPM simulations. We also introduced a novel MPM for heat transport, melting, and solidifying materials. This brings a wider range of material behaviors into reach of the already versatile MPM, which is in contrast to best-of-breed fluid, solid, or rigid-body solvers that are difficult to adapt to a wide range of materials.

Extending the MPM required several technical contributions. We introduced a dilational/deviatoric splitting of the constitutive model and show that an implicit treatment of the Eulerian evolution of the dilational part can be used to simulate arbitrarily incompressible materials. Furthermore, we showed that this treatment reduces to a parabolic equation for moderate compressibility and an elliptic, Chorin-style projection at the incompressible limit. Since projections are naturally done on Marker-And-Cell (MAC) grids, we devised a staggered-grid MPM method. Lastly, to generate varying material parameters, we adapted a heat-equation solver to a material point framework. The heat solver captures the underlying thermodynamics and alters mechanical parameters. The

method is implicit and capable of simulating nearly incompressible materials using a Chorin-like projection solve. Hence, we have broadened the range of materials that MPM can handle, and demonstrated the greater scope of our technique with several compelling melting and solidifying examples.

Our methods were shown to be computationally efficient, numerically stable, and physically accurate. These features enabled them to be directly applied within the visual effects industry. A large portion of the work on simulating multiphase physics (such as the melting and solidifying of materials) and the unified simulation of energetic matter (such as the dynamics of incompressible fluid and granular materials) has been employed in Disney's feature animations. Our research has contributed to stunning levels of visual realism for 3D animations and movie special effects, as well as for use by the interactive games industry.

Advancing the digital entertainment industry with mathematical and computational technology has had a significant positive impact on a great many people's lives. However, the physics-based simulation methods developed in this thesis can inspire other important multidisciplinary applications. In particular, our work on the unified simulation of solids and fluids can be used to create integrated numerical and graphical physics-based models of human tissue, such as to realize a realistic biomechanical model of the liver and its associated fluidic systems with associated injury and surgical procedure toolkits. Furthermore, our work on the simulation of the cutting of deformable objects [Wang et al., 2014] promises to impact on the future development of virtual surgery simulators.[1] It can potentially become a framework for training surgeons to remotely perform robotically-mediated surgeries.

[1] See, e.g., Xbox + math = virtual surgery, UCLA. https://www.youtube.com/watch?v=lT9IWPCaDNk

APPENDIX A

An Optimization-Based Integrator

Practical time steps in state-of-the-art simulators typically rely on Newton's method to solve large systems of nonlinear equations. In practice, this works well for small time steps but is unreliable at large time steps at or near the frame rate, particularly for difficult or stiff simulations. In this appendix, we show that recasting the backward Euler method as a minimization problem allows Newton's method to be stabilized by standard optimization techniques with some novel improvements of our own. The resulting solver is capable of solving even the toughest simulations at the 24 Hz frame rate and beyond. We show how simple collisions can be incorporated directly into the solver through constrained minimization without sacrificing efficiency. We show that these techniques improve the behavior of MPM simulations.

A.1 Introduction

The most commonly used time integration schemes for graphics applications are implicit methods. Among these, the backward Euler method [Baraff and Witkin, 1998; Hirota et al., 2001; Volino and Magnenat-Thalmann, 2001; Martin et al., 2011; Liu et al., 2013] or variants of the Newmark method [Kane, 1999; Bridson et al., 2002, 2003] are the most common, though even more sophisticated schemes like BDF-2 [Hauth and Etzmuss, 2001; Choi and Ko, 2005], implicit-explicit schemes [Eberhardt et al., 2000; Stern and Grinspun, 2009], or even the more exotic exponential integrators [Michels

et al., 2013] have received consideration. Integrators have been the subject of comparison before (e.g., [Hauth and Etzmuss, 2001; Volino and Magnenat-Thalmann, 2001; Parks and Forsyth, 2002]), seeking good compromises between speed, accuracy, robustness, and dynamic behavior.

These integrators require the solution to one or more nonlinear systems of equations in each time step. These systems are typically solved by some variation on Newton's method. Even the most stable simulators are typically run several time steps per 24 Hz frame of simulation. There is growing interest in running simulations at larger time steps [Su et al., 2013], so that the selection of Δt can be made based on other factors, such as damping or runtime, and not only on whether the simulator works at all. One of the major factors that limits time-step sizes is the inability of Newton's method to converge reliably with large time steps (Figures A.2, A.3, and A.4), or if a fixed number of Newton iterations are taken, the stability of the resulting simulation.

We address this by formulating our nonlinear system of equations as a minimization problem, which we demonstrate can be solved more robustly. The idea that dynamics, energy, and minimization are related has been known since antiquity and is commonly leveraged in variational integrators [Simo et al., 1992; Kane et al., 1999; Kane, 1999; Lew et al., 2004; Kharevych et al., 2006; Stern and Grinspun, 2009; Gonzalez et al., 2010]. The idea that the nonlinear system that occurs from methods like backward Euler can be formulated as a minimization problem has appeared many times in graphics in various forms [Hirota et al., 2001; Kharevych et al., 2006; Martin et al., 2011; Liu et al., 2013; Michels et al., 2013]. Kharevych et al. [2006] point out that minimization leads to a method that is both simpler and faster than the equivalent nonlinear root-finding problem, and Liu et al. [2013] show that a minimization formulation can be used to solve mass-spring systems more efficiently. Kane et al. [1999] use a minimization formulation as a means of ensuring that a solution to their nonlinear system can be found assuming one exists. Goldenthal et al. [2007] show that a minimization

formulation can be used to enforce constraints robustly and efficiently. Hirota et al. [2001] show that supplementing Newton's method with a line search greatly improves robustness. Martin et al. [2011] also show that supplementing Newton's method with a line search and a definiteness correction leads to a robust solution procedure. Following their example, we show that recasting the solution of the nonlinear systems that result from implicit time integration schemes as a nonlinear optimization problem results in substantial robustness improvements. We also show that additional improvements can be realized by incorporating additional techniques like Wolfe condition line searches which curve around collision bodies, conjugate gradient with early termination on indefiniteness, and choosing conjugate gradient tolerances based on the current degree of convergence.

This chapter covers [Gast et al., 2015], where we applied the optimization integrator approach to the MPM snow simulator. This allows us to take much larger time steps than the original method and results in a significant speedup.

A.2 Time Integration

First we consider a general Lagrangian simulation. The equations of motion for simulating solids are $\dot{\boldsymbol{x}} = \boldsymbol{v}$ and $\boldsymbol{M}\dot{\boldsymbol{v}} = \mathbf{f}$, where $\mathbf{f} = \mathbf{f}(\boldsymbol{x}, \boldsymbol{v})$ are forces. As is common in graphics, we assume $\boldsymbol{M}$ is a diagonal lumped-mass matrix. Since we are interested in robustness and large time steps, we follow a backward Euler discretization. This leads to $\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n}{\Delta t} = \boldsymbol{v}^{n+1}$ and $\boldsymbol{M}\frac{\boldsymbol{v}^{n+1}-\boldsymbol{v}^n}{\Delta t} = \mathbf{f}^{n+1} = \mathbf{f}(\boldsymbol{x}^{n+1}, \boldsymbol{v}^{n+1})$. Eliminating $\boldsymbol{v}^{n+1}$ using the first equation yields $\boldsymbol{M}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2} = \mathbf{f}(\boldsymbol{x}^{n+1}, \frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n}{\Delta t})$, which is a nonlinear system of equations in the unknown positions $\boldsymbol{x}^{n+1}$. This system of nonlinear equations is normally solved with Newton's method. If we define $\mathbf{h}(\boldsymbol{x}^{n+1}) = \boldsymbol{M}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2} - \mathbf{f}(\boldsymbol{x}^{n+1}, \frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n}{\Delta t})$, then our nonlinear problem is one of finding a solution to $\mathbf{h}(\boldsymbol{x}) = \mathbf{0}$. To do this, one would start with an initial guess $\boldsymbol{x}^{(0)}$,

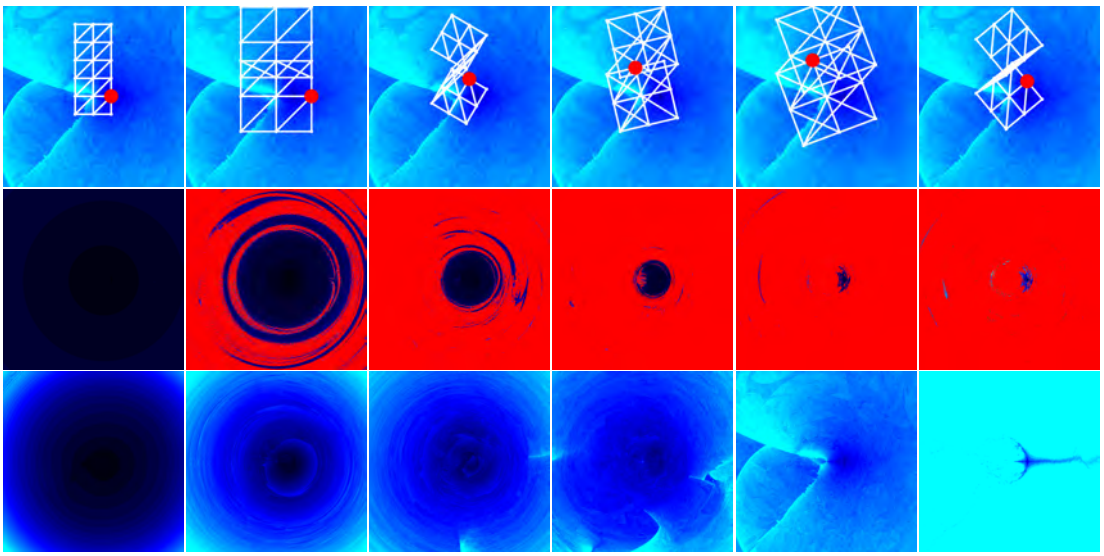

Figure A.1: *Convergence of Newton's method (middle) and our stabilized optimization formulation (bottom) for a simple 36-dof simulation in 2D. The initial configuration (top) is parameterized in terms of a pixel location, with the rest configuration occurring at* $(\frac{3}{5}, \frac{1}{2})$*. The initial velocity is zero, and 1 time step is attempted. Time steps are (left to right) 170, 40, 20, 10, and 1 steps per 24 Hz frame, with the rightmost image being* $\Delta t = 1\,s$*. Color indicates convergence in 0 iterations (black), 15 iterations (blue), 30 or more iterations (cyan), or failure to converge in 500 iterations (red). Note that Newton's method tends to converge rapidly or not at all, depending strongly on problem difficulty and the initial guess.*

such as the value predicted by forward Euler. This estimate is then iteratively improved using the update rule $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - \left(\frac{\partial \mathbf{h}}{\partial \boldsymbol{x}}(\boldsymbol{x}^{(i)})\right)^{-1}\mathbf{h}(\boldsymbol{x}^{(i)})$. Each step requires the solution of a linear system, which is usually symmetric and positive definite and is solved with a Krylov solver such as conjugate gradient or MINRES.

If $\mathbf{h}(\boldsymbol{x})$ is well-behaved and the initial guess is sufficiently close to the solution, Newton's method will converge very rapidly (quadratically). If the initial guess is not close enough, Newton's method may converge slowly or not at all. For small enough time steps, the forward and backward Euler time steps will be very similar (they differ by $O(\Delta t^2)$), so a good initial guess is available. For large time steps, forward Euler will be unstable, so it will not provide a good initial guess. Further, as the time step grows, Newton's method may become more sensitive to the initial guess (Figure A.1). The

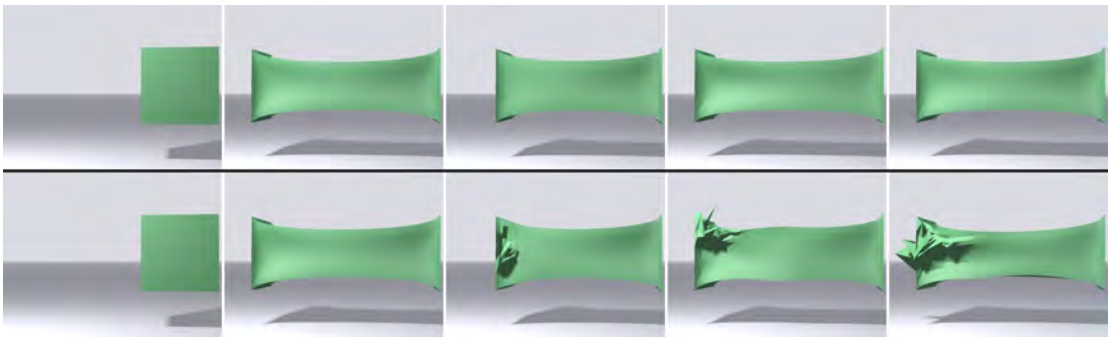

Figure A.2: *A cube being stretched and then given a small compressive pulse, shown with our method (top) and the standard Newton's method (bottom). Both simulations were run with 1 time step per 24 Hz frame. In this simulation, Newton's method is able to converge during the stretch phase, but a simple pulse of compression, as would normally occur due to a collision, causes it to fail to converge and never recover. Newton's method requires 5 time steps per frame to converge on this simple example.*

Figure A.3: *A cube being stretched: Initial configuration (left), our method at* $t = 0.4\,s$ *and* $t = 3.0\,s$ *(middle), and standard Newton's method at* $t = 0.4\,s$ *and* $t = 3.0\,s$ *(right). Both simulations were run with 1 time step per 24 Hz frame. Newton's method requires 3 time steps per frame to converge on this simple example.*

result is that Newton's method will often fail to converge if the time step is too large. Figures A.2, A.3, and A.4 show examples of simulations that ought to be routine but where Newton's method fails to converge at $\Delta t = 1/24\,\mathrm{s}$.

Sometimes only one or a small fixed number of Newton steps are taken rather than trying to solve the nonlinear equation to a tolerance. The idea is that a small number of Newton steps is sufficient to get most of the benefit from doing an implicit method while limiting its cost. Indeed, even a single Newton step with backward Euler can allow time steps orders of magnitude larger than explicit methods. Linearizing the problem only goes so far, though, and even these solvers tend to have time step restrictions for tough problems.

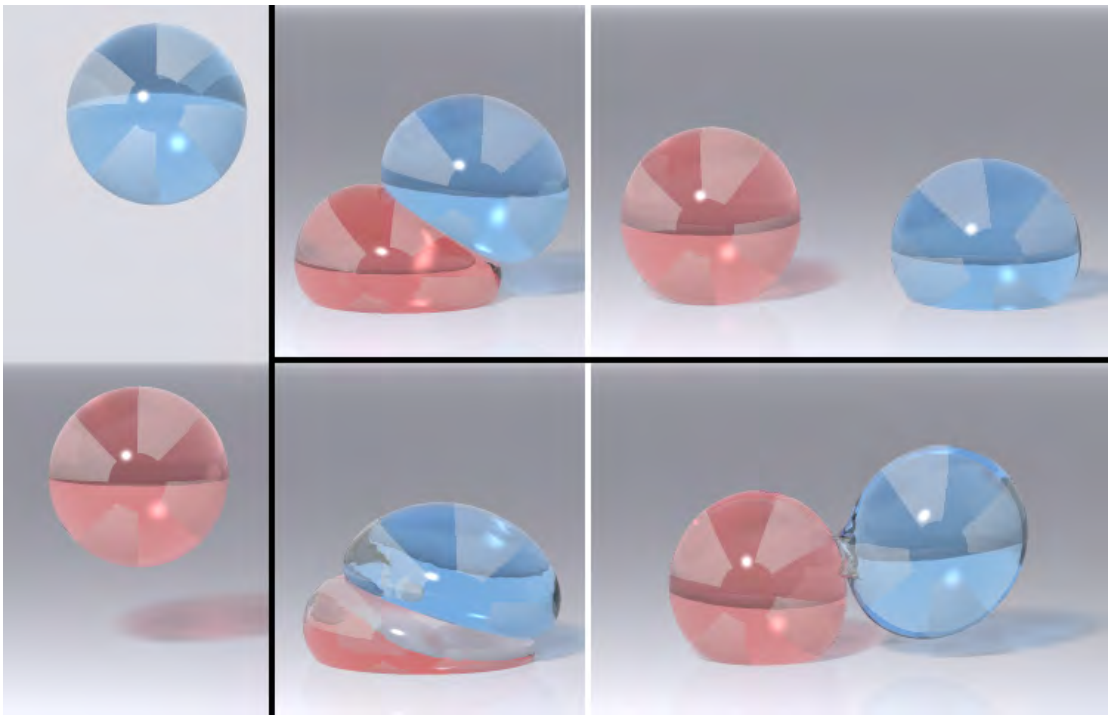

Figure A.4: *Two spheres fall and collide with one another with $\Delta t = 1/24\,s$: Initial configuration (left), our method (top), and Newton's method (bottom). Notice the artifacts caused by Newton's method not converging. Newton's method requires 6 time steps per frame to converge on this example.*

A.2.1 Minimization problem

The solution to making Newton's method converge reliably is to recast the equation-solving problem as an optimization problem, for which robust and efficient methods exist. In principle, that can always be done, since solving $\mathbf{h}(\boldsymbol{x}) = \mathbf{0}$ is equivalent to minimizing $\|\mathbf{h}(\boldsymbol{x})\|$ assuming a solution exists. This approach is not very convenient, though, since it requires a global minimum of $\|\mathbf{h}(\boldsymbol{x})\|$. Further, minimization using Newton's method would require the Hessian of $\|\mathbf{h}(\boldsymbol{x})\|$, which involves the second derivatives of our forces. The standard approach only requires first derivatives. What we really want is a quantity E that we can minimize whose second derivatives only require the first derivatives of our forces. That is, we need to *integrate* our system of nonlinear equations $\mathbf{h}(\boldsymbol{x})$. It turns out that if we require our forces to come from an energy, we can do this. This way of recasting the problem also requires that only a

local minimum be found. Most forces with symmetric force derivatives can be put into this model. We will show later how damping can also be incorporated into this model. Friction can be given an approximate potential which is valid for small Δt [Pandolfi et al., 2002]. Since our examples focus on taking larger time steps, we incorporate friction explicitly after the Newton solve.

Let Φ be the total potential energy of our internal forces (gravity is a special case, which we will address later). The potential Φ has a global minimum, which is typically its rest configuration. Then, we can write $\mathbf{h}(\boldsymbol{x}) = \boldsymbol{M}\frac{\boldsymbol{x}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2} + \frac{\partial\Phi}{\partial\boldsymbol{x}}$. We need to express this as the gradient of some scalar function $E(\boldsymbol{x})$. Letting $\hat{\boldsymbol{x}} = \boldsymbol{x}^n + \Delta t\boldsymbol{v}^n$, we have $E(\boldsymbol{x}) = \frac{1}{2\Delta t^2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^T\boldsymbol{M}(\boldsymbol{x}-\hat{\boldsymbol{x}})+\Phi$. Then, $\mathbf{h} = \frac{\partial E}{\partial\boldsymbol{x}}$ as desired. Since $\boldsymbol{M}$ is symmetric and positive definite (or at least semidefinite if scripted objects are permitted), the first term is bounded from below by zero. Since Φ is also bounded from below, E is as well. Thus, E has a global minimum. If $E(\boldsymbol{x}^{n+1})$ is smooth at its minima, then this minimum will satisfy $\frac{\partial E}{\partial\boldsymbol{x}}(\boldsymbol{x}^{n+1}) = \mathbf{0}$ or equivalently $\mathbf{h}(\boldsymbol{x}^{n+1}) = \mathbf{0}$. Note that, although we are now doing minimization rather than root finding, we are still solving the same equations. The discretization and dynamics will be the same, but the solver will be more robust.

Gravity: A graphics simulation would not be very useful without gravity. Gravity has the potential energy function $-\boldsymbol{M}\mathbf{g}^T\boldsymbol{x}$, where $\mathbf{g}$ is the gravitational acceleration vector, but this function is not bounded. An object can fall arbitrarily far and liberate a limitless supply of energy, though in practice this fall will be stopped by the ground or some other object. Adding the gravity force to our nonlinear system yields $\mathbf{h}(\boldsymbol{x}) = \boldsymbol{M}\frac{\boldsymbol{x}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2} - \boldsymbol{M}\mathbf{g} + \frac{\partial\Phi}{\partial\boldsymbol{x}}$, which can be obtained from the bounded minimization objective $E(\boldsymbol{x}) = \frac{1}{2\Delta t^2}(\boldsymbol{x} - \hat{\boldsymbol{x}} - \Delta t^2\mathbf{g})^T\boldsymbol{M}(\boldsymbol{x} - \hat{\boldsymbol{x}} - \Delta t^2\mathbf{g}) + \Phi$. A more convenient choice of E, and the one we use in practice, is obtained by simply adding the effects of gravity $\Phi_g = -\boldsymbol{M}\mathbf{g}^T\boldsymbol{x}$ into Φ. Since all choices E will differ by a constant shift, this more convenient minimization objective will also be bounded from below.

A.3 Minimization

The heart of our simulator is our algorithm for solving optimization problems, which we derived primarily from [Nocedal and Wright, 2006], though most of the techniques we apply are well-known. We begin be describing our method as it applies to unconstrained minimization and then show how to modify it to handle the constrained case.

A.3.1 Unconstrained minimization

Our optimization routine begins with an initial guess, $\boldsymbol{x}^{(0)}$. Each iteration consists of the following steps:

1. * Register active set
2. Compute gradient ∇E and Hessian $\boldsymbol{H}$ of E at $\boldsymbol{x}^{(i)}$
3. Terminate successfully if $\|\nabla E\| < \tau$
4. Compute Newton step $\Delta x = -\boldsymbol{H}^{-1}\nabla E$
5. Make sure Δx is a downhill direction
6. Clamp the magnitude of Δx to ℓ if $\|\Delta x\| > \ell$
7. Choose step size α in direction Δx using a line search
8. Take the step: $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha\Delta x$
9. * Project $\boldsymbol{x}^{(i+1)}$

Here, τ is the termination criterion, which controls how accurately the system must by solved. The length clamp ℓ guards against the possibility of the Newton step being enormous (if $\|\Delta x\| = 10^{100}$, computing $\Phi(\boldsymbol{x}^{(i)} + \Delta x)$ is unlikely to work well). Its

value should be very large. Our line search is capable of choosing $\alpha > 1$, so the algorithm is very insensitive with respect to the choice ℓ. We normally use $\ell = 10^3$. Steps beginning with * are only performed for constrained optimization and will be discussed later. A few of the remaining steps require further elaboration here.

Linear solver considerations: Computing the Newton step requires solving a symmetric linear system. The obvious candidate solver for this is MINRES that can handle indefinite systems, and indeed this will work. However, there are many tradeoffs to be made here. In contrast to a normal Newton solve, an accurate estimate for Δx is not necessary for convergence. Indeed, we would still converge with high probability if we chose Δx to be a random vector. The point of using the Newton direction is that convergence will typically be much more rapid, particularly when the superconvergence of Newton's method kicks in. (Choosing $\Delta x = -\nabla E$ leads to gradient descent, for example, which can display notoriously poor convergence rates.) When the current estimate is far from the solution, the exact Newton direction tends to be little better than a very approximate one. Thus, the idea is to spend little time on computing Δx when $\|\nabla E\|$ is large and more time when it is small. We do this by solving the system to a relative tolerance of $\min(\frac{1}{2}, \sigma\sqrt{\max(\|\nabla E\|, \tau)})$. The $\frac{1}{2}$ ensures that we always reduce the residual by at least a constant factor, which guarantees convergence. The scale σ adjusts for the fact that ∇E is not unitless (we usually use $\sigma = 1$). If our initial guess is naive, we must make sure we take at least one minimization iteration, even if ∇E is very small. Using τ here ensures that we do not waste time solving to a tiny tolerance in this case.

Conjugate gradient: One further optimization is to use conjugate gradient as the solver with a zero initial guess. If indefiniteness is encountered during the conjugate gradient solve, return the last iterate computed. If this occurs on the first step, return the right hand side. If this is done, Δx is guaranteed to be a downhill direction, though it might not be sufficiently downhill for our purposes. In practice, indefiniteness will only occur

if far from converged, in which case little time is wasted in computing an accurate Δx that is unlikely to be very useful anyway. Indeed, if the system is detectably indefinite and Δx is computed exactly; it might not even point downhill. Since we are searching for a minimum of E (even a local one), the Hessian of E will be symmetric and positive definite near this solution. (Technically, it need only be positive semidefinite, but in practice this is of little consequence.) Thus, when we are close enough to the solution for an accurate Newton step to be useful, conjugate gradient will suffice to compute it. This is very different from the normal situation, where a solver like MINRES or an indefiniteness correction are employed to deal with the possibility of indefiniteness. In the case of our solver, neither strategy is necessary, and both make the algorithm slower.

Downhill direction: Making sure Δx points downhill is fairly straightforward. If $\Delta x \cdot \nabla E < -\kappa \|\Delta x\| \|\nabla E\|$, then we consider Δx to be suitable. Otherwise, if $-\Delta x$ is suitable, we use it instead. If neither Δx nor $-\Delta x$ are suitable, then we use the gradient descent direction $-\nabla E$. Note that if the conjugate gradient strategy is used for computing the Newton direction, then $-\Delta x$ will never be chosen as the search direction at this stage. We have found $\kappa = 10^{-2}$ to work well.

Line search: For our line search procedure, we use an algorithm for computing α such that the strong Wolfe Conditions are satisfied. See [Nocedal and Wright, 2006] for details. The line search procedure guarantees that E never increases from one iteration to the next and that, provided certain conditions are met, sufficient progress is always made. One important attribute of this line search algorithm is that it first checks to see if Δx itself is a suitable step. In this way, the line search is almost entirely avoided when Newton is converging properly.

Initial guess: A good initial guess is important for efficient simulation under normal circumstances. Under low-Δt or low-stress conditions, a good initial guess is obtained by replacing $\mathbf{f}^{n+1}$ by $\mathbf{f}^n$, resulting in $\boldsymbol{M}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2} = \mathbf{f}(\boldsymbol{x}^n)$. Solving for $\boldsymbol{x}^{n+1}$

yields the initial guess $\boldsymbol{x}^{(0)} = \boldsymbol{x}^n + \Delta t \boldsymbol{v}^n + \Delta t^2 \mathbf{f}(\boldsymbol{x}^n)$. This initial guess is particularly effective under free fall, since here the initial guess is correct and no Newton iterations are required. On the other hand, this initial guess is the result of an explicit method, which will be unstable at large time steps or high stress. Under these conditions, this is unlikely to be a good initial guess and may in fact be very far from the solution. Under these situations, a better initial guess is obtained from $\boldsymbol{x}^{(0)} = \boldsymbol{x}^n + \Delta t \boldsymbol{v}^n$. In practice, we compute both initial guesses and choose the one which produces the smaller value of E. This way, we get competitive performance under easy circumstances and rugged reliability under tough circumstances.

A.3.2 Constrained minimization

We use constrained minimization for some of our collisions, which may result in a large active set of constraints, such as when an ball is bouncing on the ground. As the ball rises, constraints become deactivated. As the ball hits the ground, more constraints become activated. The change in the number of active constraints from iteration to iteration may be quite significant. This would render a traditional active set method impractical, since constraints are activated or deactivated one at a time. Instead, we use the gradient-projection method as our starting point, since it allows the number of active constraints to change quickly. The downside to this choice is that its reliance on the ability to efficiently project to the feasible region limits its applicability to simple collision objects.

Projections: Let $P(\boldsymbol{x})$ be the projection that applies P_{bp} to $\boldsymbol{x}_p$ for all body-particle pairs (b, p) that are labeled as active or are violated ($\phi_b(\boldsymbol{x}_p) < 0$). Note that pairs such that $\phi_b(\boldsymbol{x}_p) = 0$ (as would be the case once projected) are considered to be touching but not violated. The iterates $\boldsymbol{x}^{(i)}$ obtained at the end of each Newton step, as well as the initial guess, are projected with P.

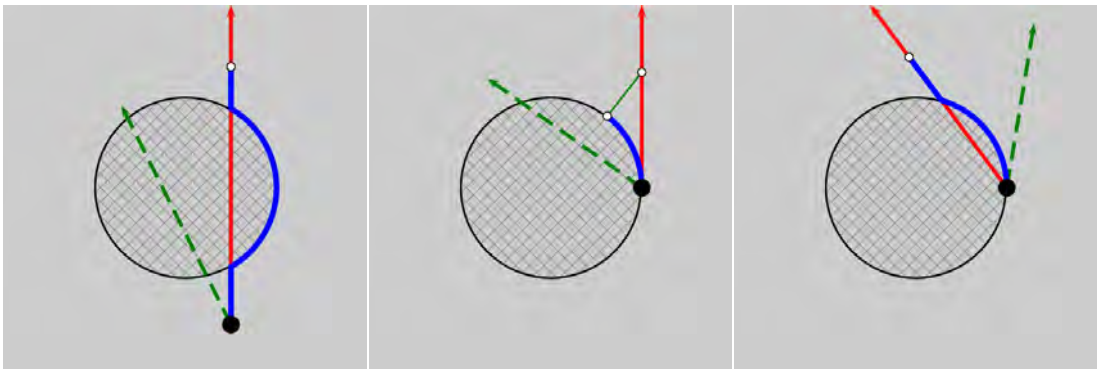

Figure A.5: *Line search showing the gradient descent direction (green), Newton direction (red), and effective line search path (blue). The constraint is initially feasible (left), active (middle), and touching but inactive (right). Constraints are projected if violated or active, but only inactive constraints may separate.*

Register active set: Let E' be the objective that would be computed in the unconstrained case. The objective function for constrained optimization is $E(\boldsymbol{x}) = E'(P(\boldsymbol{x}))$. Compute the gradient $\nabla E'$. Constraints that are touching and for which $\nabla E' \cdot \nabla \phi_b \geq 0$ are labeled as active for the remainder of the Newton step. All others are labeled as inactive. No constraint should be violated at this stage. Note that $E'(\boldsymbol{x}^{(i)}) = E(\boldsymbol{x}^{(i)})$ is true before and after every Newton step, since constraints are never violated there.

Curved paths: Note that configurations are always projected to the feasible region before E is computed. One may interpret this as performing line searches along curved paths, as illustrated is Figure A.5. When the unprojected line search curve passes through the medial axis of an object, it is possible for the search curve to be disconnected. This causes a discontinuity in the energy as seen from the line search. If the line search does not stop at the discontinuity, the discontinuity has no effect. If it does, the constraint causing the discontinuity will be active (in which case the discontinuity is projected out) or separating (in which case we move away from the discontinuity) in the next Newton step. Thus a disconnected search curve is not a problem for our method.

Derivatives: Note also that E must be differentiated twice, and that involves differen-

tiating the projection function P twice. Since P depends on the first derivatives of ϕ_b, the Hessian $\boldsymbol{H}$ of E would seem to require third derivatives. We note, however, that the only occurrence of the third derivative of ϕ_b occurs multiplied by ϕ_b. Since $\boldsymbol{H}$ is used only at the beginning of the Newton step when the configuration is feasible, $\phi_b(\boldsymbol{x}_p) = 0$ or P_{bp} is the identity function. The third derivative term is zero either way, so only the second derivatives of ϕ_b are required.

A.3.3 Practical considerations

There are a few matters of practicality that are worth mentioning regarding the effective use of this method. The most important of these is that the method does not tolerate discontinuities in E, not even very minute ones, except under some special circumstances that we mention below. In practice, what tends to happen is that a line search encounters a discontinuity in E, where E rises abruptly. The line search dutifully advances the configuration right up to location of this discontinuity. If in the next Newton iteration the descent direction points into the discontinuity, no progress can be made. The solver is stuck. Discontinuities in ∇E can also cause problems and are impossible to avoid in general. As long as these kinks are not valleys of E, they are fine. Thus, the corotated constitutive model, though not completely unusable with this solver, is ill-advised (the fixed variant [Stomakhin et al., 2012] has no such valleys and is fine). In practice, we have only encountered problems when evaluating self-collision models. The self-collision model we propose works well with the method.

The second practical consideration is that E can be somewhat noisy. This is particularly true with forces that involve an SVD, since its computation involves a balance between speed and accuracy. If the Newton tolerance τ is set too low, the solver will be forced to optimize an objective E where the actual change in E is hidden by the noise. Even with our noisy SVD, we found there is typically at least a three order-of-magnitude range between the largest value of τ below which no change in output is visually observed

and the smallest value above which E is not too noisy to optimize reliably. If we make the E computation robust, E can be optimized down to roundoff level.

Another practical consideration is that occasionally very large changes in the configuration are considered by the line search. For most forces, this is of little consequence. For self-collisions, however, this poses a major performance hazard. We note that when this occurs, the other components of E become very large too. We first compute all contributions to E except self-collisions. Since our self-collision potential has a global minimum of zero, the real E will be at least as large as the estimate. If this partial E is larger than $E(\boldsymbol{x}^{(i)})$, we do not compute self-collisions at all. While this presents a discontinuity in E to the optimizer, it is safe to do so under these conditions, since the optimizer will avoid the large value in E by taking a smaller step along the search line.

A.4 Forces

Our formulation is fairly insensitive to the underlying forces, provided it has a continuous potential energy function. We use five forces in our simulations. The simplest of these is gravity, which we addressed in Section A.2.1. We also employ a hyperelastic constitutive model (Section A.4.1), a Rayleigh damping model (Section A.4.2), and two collision penalty force models (Sections A.5.2 and A.5.3).

A.4.1 Elastic

A suitable hyperelastic constitutive model must have a few key properties to be suitable for this integrator. The most important is that it must have a potential energy function defined everywhere, and this function must be continuous. The constitutive model must be well-defined for any configuration, including configurations that are degenerate or inverted. This is true even if objects do not invert during the simulation, since the

minimization procedure may still encounter such states. Examples of suitable constitutive models are those defined by the corotated hyperelasticity energy [Schmedding and Teschner, 2008; Zhu et al., 2010b; Muller and Gross, 2004; Etzmuss et al., 2003; Chao et al., 2010; McAdams et al., 2011] (but see Section A.3.3), and the fixed corotated hyperelasticity variant [Stomakhin et al., 2012]. Stress-based extrapolated models [Irving et al., 2004; Teran et al., 2005] are unsuitable due to the lack of a potential energy function in the extrapolated regime, but energy-based extrapolation models [Stomakhin et al., 2012] are fine. We use the fixed corotated variant [Stomakhin et al., 2012] for all of our simulations for its combination of simplicity and robustness.

A.4.2 Damping

At first, one might conclude that requiring a potential energy may limit our method's applicability, since damping forces cannot be defined by a potential energy function. A very simple damping model is given by $\mathbf{f} = -k\boldsymbol{M}\boldsymbol{v}^{n+1}$. Eliminating the velocity from the equation yields $\mathbf{f}(\boldsymbol{x}^{n+1}) = -k\boldsymbol{M}\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n}{\Delta t}$, where $k > 0$. The scalar function $\Phi(\boldsymbol{x}^{n+1}) = \frac{k}{2\Delta t}(\boldsymbol{x}^{n+1}-\boldsymbol{x}^n)^T\boldsymbol{M}(\boldsymbol{x}^{n+1}-\boldsymbol{x}^n)$ has the necessary property that $\mathbf{f} = -\frac{\partial\Phi}{\partial\boldsymbol{x}}$. Note that this Φ looks very similar to our inertial term in E, and it is similarly bounded from below. That this Φ is not a real potential energy function is evident from its dependence on $\boldsymbol{x}^n$ and Δt, but it is nevertheless suitable for use in our integrator. This simple drag force is not very realistic, though, so we do not use it in our simulations.

A more realistic damping force is Rayleigh damping. Let ψ be an elastic potential energy function. The stiffness matrix corresponding to this force is $-\frac{\partial^2\psi}{\partial\boldsymbol{x}\partial\boldsymbol{x}}$, and the Rayleigh damping force is $\mathbf{f} = -k\left(\frac{\partial^2\psi}{\partial\boldsymbol{x}\partial\boldsymbol{x}}(\boldsymbol{x}^{n+1})\right)\boldsymbol{v}^{n+1}$. This integrates to $\Phi_c = \frac{k}{\Delta t}\left((\boldsymbol{x}^{n+1}-\boldsymbol{x}^n)^T\frac{\partial\psi}{\partial\boldsymbol{x}} - \psi\right)$. This candidate Φ_c has at least two serious problems. The first is that second derivatives of Φ_c involve third derivatives of ψ. The second is that $\frac{\partial^2\psi}{\partial\boldsymbol{x}\partial\boldsymbol{x}}$ may be indefinite, in which case the damping force may not be entirely dissipative. Instead, we approximate Rayleigh damping with a lagged version. Let $\mathbf{D} = \frac{\partial^2\psi}{\partial\boldsymbol{x}\partial\boldsymbol{x}}(\boldsymbol{x}^n)$.

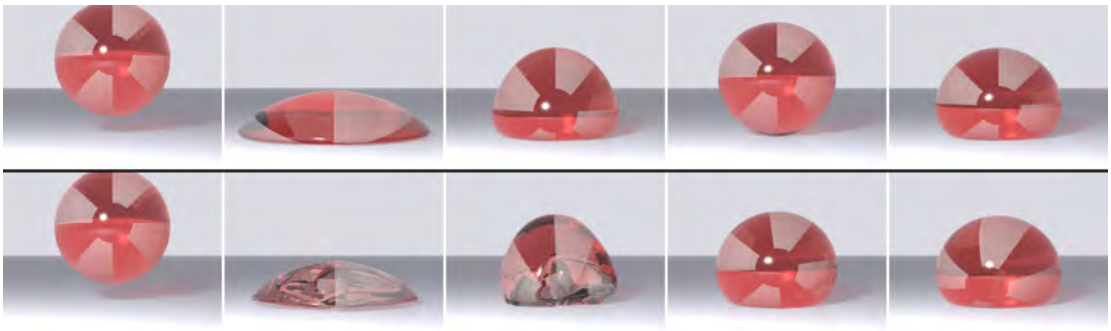

Figure A.6: *Sphere dropping hard on the ground with $\Delta t = 1/24\,s$ with constraint collisions (top) and collisions as a post-process (bottom). Penalty collisions produce a result very similar to constraint collisions, though some penetration with the ground occurs. Note that the post-processing approach leads to inversion during recovery from the collision.*

Since $\mathbf{D}$ does not depend on $\boldsymbol{x}^{n+1}$, the lagged Rayleigh damping force $\mathbf{f} = -k\mathbf{D}\boldsymbol{v}^{n+1}$ leads to $\Phi_d = \frac{k}{2\Delta t}(\boldsymbol{x}^{n+1} - \boldsymbol{x}^n)^T\mathbf{D}(\boldsymbol{x}^{n+1} - \boldsymbol{x}^n)$. This solves the first problem, since the second derivative of Φ_d is just $\frac{k}{\Delta t}\mathbf{D}$. Since $\mathbf{D}$ is not being differentiated, it is safe to modify it to eliminate indefiniteness as described in [Teran et al., 2005; Stomakhin et al., 2012]. This addresses the second problem. We did not use the damping model found in [Kharevych et al., 2006], which uses $\psi(\boldsymbol{x}^{n+1})$ with $\boldsymbol{x}^n$ used as the rest configuration, because it is not defined when $\boldsymbol{x}^n$ is degenerate.

A.5 Collisions

Collisions are a necessary part of any practical computer graphics simulator. The simplest approach to handling collisions is to process them as a separate step in the time integration scheme. This works well for small time steps, but it causes problems when used with large time steps as seen in Figure A.4. Such arrangement often leads to the collision step flattening objects to remove penetration and the elastic solver restoring the flattened geometry by pushing it into the colliding object. To get around this problem, the backward Euler solver needs to be aware of collisions. A well-tested strategy for doing this is to use penalty collisions, and we do this for two of our three collision

processing techniques.

A.5.1 Object collisions as constraints

Our first collision processing technique takes advantage of our minimization framework to treat collisions with non-simulated objects as inequality constraints. Treating collisions or contacts as constraints is not new and in fact forms the basis for LCP formulations such as [Kaufman et al., 2008; Gascon et al., 2010]. Unlike LCP formulations, however, our formulation does not attempt to be as complete and as a result can be solved about as efficiently as a simple penalty formulation.

Our constraint collision formulation works reliably when the level set is known analytically. This limits its applicability to analytic collision objects. While this approach is feasible only under limited circumstances, these circumstances occur frequently in practice. When this approach is applicable, it is our method of choice, since it produces better results (e.g., no interpenetration) for similar cost. When this formulation is not applicable, we use a penalty collision formulation instead.

We begin by representing our collision objects (indexed with b) by a level set, which we denote ϕ_b to avoid confusion with potential energy. By convention, $\phi_b(\boldsymbol{x}) < 0$ for points $\boldsymbol{x}$ in the interior of the collision object b. Our collision constraint is simply that $\phi_b(\boldsymbol{x}_p^{n+1}) \geq 0$ for each simulation particle p and every constraint collision object b. With such a formulation, we can project a particle at $\boldsymbol{x}_p$ to the closest point $\boldsymbol{x}'_p$ on the constraint manifold using $\boldsymbol{x}'_p = P_{bp}(\boldsymbol{x}_p) = \boldsymbol{x}_p - \phi_b(\boldsymbol{x}_p)\nabla\phi_b(\boldsymbol{x}_p)$. We show how to solve the resulting minimization problem in Section A.3.2.

We apply friction after the Newton solve. The total collision force felt by particles is $\nabla E'(\boldsymbol{x}^{n+1}) - \nabla E(\boldsymbol{x}^{n+1}) = \nabla E'(\boldsymbol{x}^{n+1}) - \nabla E'(P(\boldsymbol{x}^{n+1}))$ (Section A.3.2 for the definition of E'). Only collision pairs that are active at the end of the minimization will be applying such forces. We use the level set's normal and the collision force to apply

Coulomb friction to colliding particles.

Our constraint collision formulation is not directly applicable to grid-based level sets, since we assume that $P_{bp}(P_{bp}(\boldsymbol{x}_p)) = P_{bp}(\boldsymbol{x}_p)$ and $P_{bp}(x)$ is continuous. Continuity of $P_{bp}(x)$ can be achieved, for example, with C^1 cubic spline level set interpolation. However, it will not generally be true that $P_{bp}(P_{bp}(\boldsymbol{x}_p)) = P_{bp}(\boldsymbol{x}_p)$. Alternatively, the projection routine can be modified to iterate the projection to convergence, but then continuity is lost.

A.5.2 Object penalty collisions

When a collision object is not analytic, as will normally be the case for characters for instance, we use a penalty formulation instead. As in the constraint formulation, we assume our collision object is represented by a level set ϕ_b. The elastic potential energy $\Phi_{bp}(\boldsymbol{x}_p)$ of our penalty force is $\Phi_{bp}(\boldsymbol{x}) = 0$ if $\phi_b(\boldsymbol{x}_p) > 0$ and $\Phi_{bp}(\boldsymbol{x}_p) = k\phi_b(\boldsymbol{x}_p)^3$ otherwise. Since Φ_{bp} is a potential energy, we must differentiate it twice for our solver. It is important to compute the derivatives of ϕ_b exactly by differentiating the interpolation routine rather than approximating them using central differences. While a C^1 cubic spline interpolation is probably a wiser interpolation strategy since it would avoid the energy kinks that may be caused by a piecewise linear encoding of the level set, we found linear interpolation to work well, too, and we use linear interpolation in our examples.

As in the constraint case, we apply friction after the Newton solve. The total collision force felt by a particle due to object penalty collisions is obtained by evaluating the penalty force at $\boldsymbol{x}^{n+1}$. We compute the component of the discrete acceleration $\frac{\boldsymbol{x}^{n+1}-\boldsymbol{x}^n-\Delta t\boldsymbol{v}^n}{\Delta t^2}$ perpendicular to the collision force and apply Coulomb friction in the opposite direction to the colliding particle.

A.5.3 Penalty self-collisions

We detect self-collisions by performing point-tetrahedron inclusion tests, which we accelerate with a bounding box hierarchy. If a point is found to be inside a tetrahedron but not one of the vertices of that tetrahedron, then we flag the particle as colliding.

Once we know a particle is involved in a self collision, we need an estimate for how close the particle is to the boundary. If this particle has collided before, we use the primitive it last collided with as our estimate. Otherwise, we compute the approximate closest primitive in the rest configuration using a level set and use the current distance to this surface element as an estimate.

Given this upper bound estimate of the distance to the boundary, we perform a bounding box search to conservatively return all surface primitives within that distance. We check these candidates to find the closest one. Now we have a point-primitive pair, where the primitive is the surface triangle, edge, or vertex that is closest to the point being processed. Let d be the square of the point-primitive distance. The penalty collision energy for this point is $\Phi = kd\sqrt{d+\epsilon}$, where ϵ is a small number (10^{-15} in our case) to prevent the singularities when differentiating. Note that this penalty function is approximately cubic in the penetration depth. This final step is the only part that must be differentiated.

As with the other two collision models, we apply friction after the Newton solve. In the most general case, a point n_0 collides with a surface triangle with vertices n_1, n_2, and n_3. As with the object penalty collision model, collision forces are computed by evaluating $\Phi(\boldsymbol{x}^{n+1})$ and its derivative. The force applied to n_0 is denoted $\mathbf{f}$; its direction is taken to be the normal direction $\mathbf{n}$. The closest point on the triangle to n_0 has barycentric weights w_1, w_2, and w_3. Let $w_0 = -1$ for convenience. Let $\mathbf{Q} = \mathbf{I} - \mathbf{n}\mathbf{n}^T$. If we apply a tangential impulse $\mathbf{Q}\mathbf{j}$ to these particles, their new velocities will be $\hat{v}_{n_i} = v_{n_i} + w_i m_{n_i}^{-1}\mathbf{Q}\mathbf{j}$, and total kinetic energy will be $KE = \sum_{n=0}^{3} \frac{1}{2} m_{n_i} \hat{v}_{n_i}^T \hat{v}_{n_i}$. We

want to minimize this kinetic energy to prevent friction from causing instability. Since M is positive definite, we see that KE is minimized when $\nabla KE = \mathbf{Q}\overline{v} + \overline{m}^{-1}\mathbf{Q}\mathbf{j} = 0$, where $\overline{v} = \sum_{n=0}^{3} w_i v_{n_i}$ and $\overline{m}^{-1} = \sum_{n=0}^{3} w_i m_{n_i}^{-1} w_i$. Thus if we let $\mathbf{j} = -\overline{m}\mathbf{Q}\overline{v}$ then $\nabla KE = 0$, and $\mathbf{Q}\mathbf{j} = \mathbf{j}$. If $\|\mathbf{j}\| < \mu\|\mathbf{f}\|$, then we choose $\mathbf{j}' = \mathbf{j}$ as our friction impulse. Otherwise, $\mathbf{j}' = \mu\|\mathbf{f}\|\frac{\mathbf{j}}{\|\mathbf{j}\|}$. Finally, the new velocities are $v'_{n_i} = v_{n_i} + w_i m_{n_i}^{-1}\mathbf{j}'$. Note that all three friction algorithms decrease kinetic energy but do not modify positions, so none of them can add energy to the system, and thus stability ramifications are unlikely even though friction is applied explicitly. This approach to friction can have artifacts, however, since friction will be limited to removing kinetic energy from colliding particles. This limits the amount of friction that can be applied at large time steps. An approach similar to the one in [Kaufman et al., 2008] that uses successive Quadratic Programming solves could possibly be applied to eliminate these artifacts. However Zheng and James [2011] found existing large-scale sparse QP solvers to be insufficiently robust, and thus we did not use this method.

A.6 Accelerating the MPM

In this section we describe the application of this optimization approach to the MPM snow simulation. The approach discussed in [Stomakhin et al., 2013] and Section 2.2 used an energy-based formulation to facilitate a semi-implicit treatment of the MPM. While this leads to a significant time step improvement over more standard explicit treatments, it still requires a small time step in practice to remain stable. We show how to modify their original formulation so that we are able to take time steps on the order of the CFL condition. We also provide an improved treatment of collisions with solid bodies that naturally handles them as constraints in the optimization. Although the optimization solve is for grid velocities, we show that a backward Euler (rather than forward Euler) update of particle positions in the grid-based velocity field automatically guarantees no particles penetrate solid bodies. In addition to the significantly improved

stability, we demonstrate in Section 2.3 that in many cases a worthwhile speedup can be obtained with our new formulation.

In Section 4.1 of [Stomakhin et al., 2013], the original method is broken down into 10 steps. We retain steps 1-2 unaltered. Steps 3-6 are replaced with our optimization formulation described in Section A.6.1. Steps 7-8 are unaltered. Step 9 is omitted entirely, and step 10 is replaced by the implicit update in Section A.6.2.

A.6.1 Optimization formulation

The primary modification that we propose is to use the optimization framework in place of the original solver. For this, we must formulate their update in terms of an optimization objective E. The original formulation defined the potential energy $\Phi(\boldsymbol{x}_\mathbf{i})$ conceptually in terms of the grid node locations $\boldsymbol{x}_\mathbf{i}$. Here we use the index $\mathbf{i}$ to refer to grid node indices. Their grid is a fixed Cartesian grid and never moves, and they solve for $\boldsymbol{v}_\mathbf{i}^{n+1}$. We will follow the same conceptual formulation here. This leads to the objective $E(\boldsymbol{v}_\mathbf{i}) = \sum_\mathbf{i} \frac{1}{2} m_\mathbf{i} \|\boldsymbol{v}_\mathbf{i} - \boldsymbol{v}_\mathbf{i}^n\|^2 + \Phi(\boldsymbol{x}_\mathbf{i}^n + \Delta t \boldsymbol{v}_\mathbf{i})$, where $m_\mathbf{i}$ is the mass assigned to grid index $\mathbf{i}$. Our final $\boldsymbol{v}_\mathbf{i}^{n+1}$ is computed so that $E(\boldsymbol{v}_\mathbf{i}^{n+1})$ is minimized. We solve this minimization problem as in Section A.3. Note that we apply plasticity explicitly as in the original formulation.

Using larger time steps causesour linear systems to become slower to solve. In the case of the MPM, we found it beneficial to use the diagonal preconditioner $\mathbf{L}_\mathbf{ii} = \sum_p \operatorname{diag}(m_p w_{\mathbf{i}p} \mathbf{I} + \Delta t^2 V_p^0 \boldsymbol{H})$, where $\boldsymbol{H} = (\lambda_p + \mu_p) \nabla w_{\mathbf{i}p} \nabla w_{\mathbf{i}p}^T + \mu_p \nabla w_{\mathbf{i}p}^T \nabla w_{\mathbf{i}p} \mathbf{I}$. This preconditioner approximates the diagonal of the stiffness matrix at the rest configuration. This works well since snow is unable to deform much without hardening or fracturing. We use an approximation to the diagonal, rather than the exact diagonal, because we never explicitly form the matrix. This approximation suffices for preconditioning and is more efficient.

The original method performed solid body collisions while computing new grid velocities. We treat body collisions using constraints in our optimization problem. We assume sticking collisions and let $P(\boldsymbol{v}_{\mathbf{i}}) = \mathbf{0}$ for all grid nodes $\mathbf{i}$ that lie inside a collision object. Note that we do not permit separation during optimization, though separation may occur during other steps in the algorithm.

A.6.2 Particle position update

One of the difficulties with running the method of [Stomakhin et al., 2013] with larger time steps is the particle-based solid body collisions. They were needed under the old formulation to prevent settling into the ground, but at the same time they cause bunching of particles at collision objects. These problems are exacerbated at larger time steps, and another approach is required. Instead, we show that altering the way we update particle positions can avoid the need for a separate particle collision step.

For each particle position $\boldsymbol{x}_p$ we solve the backward Euler update equation $\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \boldsymbol{v}(\boldsymbol{x}_p^{n+1})$, where $\boldsymbol{v}(\boldsymbol{x}_p^{n+1}) = \sum_{\mathbf{i}} \boldsymbol{v}_{\mathbf{i}}^{n+1} N_{\mathbf{i}}^h(\boldsymbol{x}_p)$ is the interpolated grid velocity at the particle location $\boldsymbol{x}_p^{n+1}$. These updates are independent per particle and so are relatively inexpensive. A solution to this backward Euler equation always exists nearby. Note that pure PIC velocities are used in the particle position updates. While a combination of FLIP/PIC is still stored on particles (to avoid excessive dissipation in subsequent transfer to grid), PIC velocities for position updates lead to more stable behavior.

The motivation for our modification can be best understood in the case of sticking collisions. Inside a collision object, we will have $\boldsymbol{v}_{\mathbf{i}}^{n+1} = \mathbf{0}$ due to the collision constraints imposed during optimization. If we then assume that we will interpolate $\boldsymbol{v}(\boldsymbol{x}_p^{n+1}) = \mathbf{0}$ here, then we can see from $\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \boldsymbol{v}(\boldsymbol{x}_p^{n+1})$ that $\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n$. Note that if a particle ends up inside the collision object, then it must have already been there. Thus,

it is not possible for particles to penetrate collision objects. In our implementation, $\boldsymbol{v}(\boldsymbol{x}_p^{n+1}) = \mathbf{0}$ will only be true if we are slightly inside collision objects, but in practice this procedure actually stops particles slightly outside collision objects.

We solve this equation with Newton's method. Since Newton's method need not converge, some care is required, though in practice nothing as sophisticated as Section A.3 is needed. We always use the Newton direction but repeatedly halve the length of the Newton step until the objective $E = \|\boldsymbol{x}_p^{n+1} - \boldsymbol{x}_p^n - \Delta t \boldsymbol{v}(\boldsymbol{x}_p^{n+1})\|$ no longer increases. (If halving the step size 14 times does not suffice, we take the reduced step anyway.) Typically, only one Newton step is required for convergence. We have never observed this to fail.

We use a quadratic spline rather than the cubic of the original formulation to reduce stencil width and improve the effectiveness of the modified position update. That is, we let $N(x) = \frac{3}{4} - x^2$ for $|x| < \frac{1}{2}$, $N(x) = \frac{1}{2}x^2 - \frac{3}{2}|x| + \frac{9}{8}$ for $\frac{1}{2} \leq |x| < \frac{3}{2}$, and $N(x) = 0$ otherwise. Using a quadratic stencil also has the advantage of being more efficient. We do not use a linear spline since it is not smooth enough for Newton's method to be effective in the particle position update.

Since the MPM involves a grid, we limit our time step so that particles do not travel more that one grid spacing per time step. That is, we choose Δt so that $\nu \frac{\Delta x}{\Delta t} \geq \max_p \|\boldsymbol{v}_p^n\|$ for some $\nu < 1$. We chose $\nu = 0.6$ for our examples. Although the time step restriction is computed based on $\boldsymbol{v}_p^n$ rather than $\boldsymbol{v}_p^{n+1}$, this suffices in practice.

A.7 Simulation results

We begin by demonstrating how robust our solver is by considering the two most difficult constitutive model tests we are aware of—total randomness and total degeneracy. The attributes that make them tough constitutive model tests also make them tough

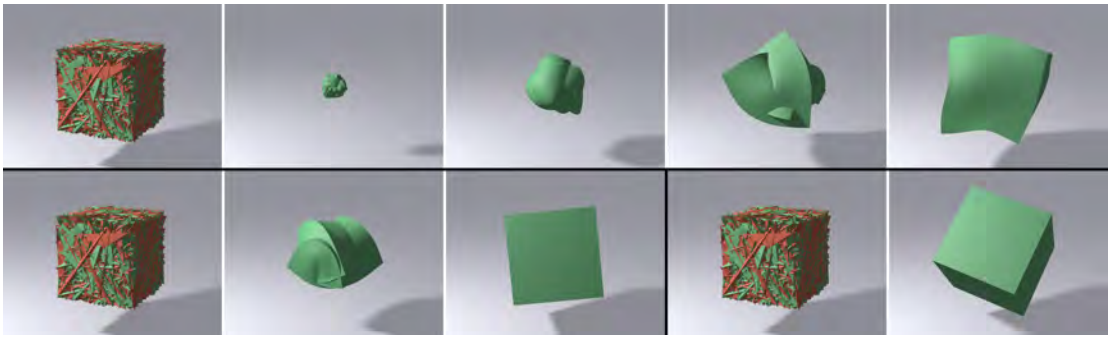

Figure A.7: *Random test with* $65 \times 65 \times 65$ *particles simulated with* $\Delta t = 1/24\,s$ *for three stiffnesses: Low stiffness recovering over 100 time steps (top), medium stiffness recovering over 40 time steps (bottom left), and high stiffness recovering in a single time step (bottom right). The red tetrahedra are inverted, while the green are uninverted.*

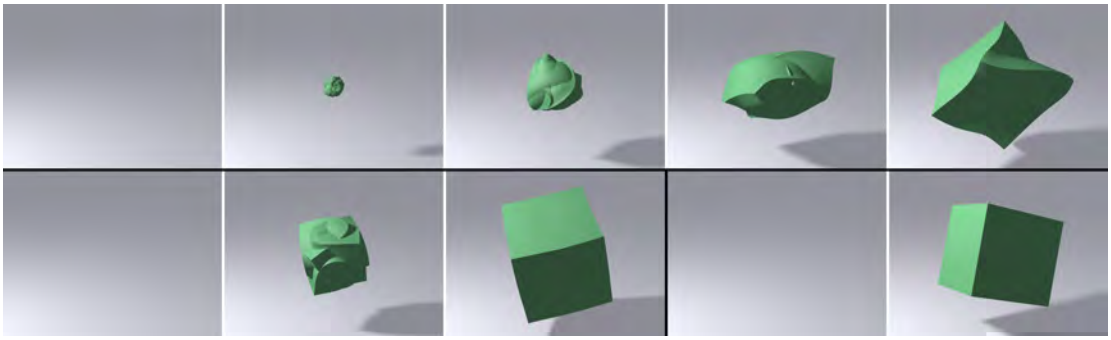

Figure A.8: *Point test with* $65 \times 65 \times 65$ *particles simulated with* $\Delta t = 1/24\,s$ *for three stiffnesses: Low stiffness recovering over 120 time steps (top), medium stiffness recovering in 5 time steps (bottom left), and high stiffness recovering in a single time step (bottom right).*

Figure A.9: *A torus falls on the ground (constraint collisions) and collides with itself (penalty collisions).*

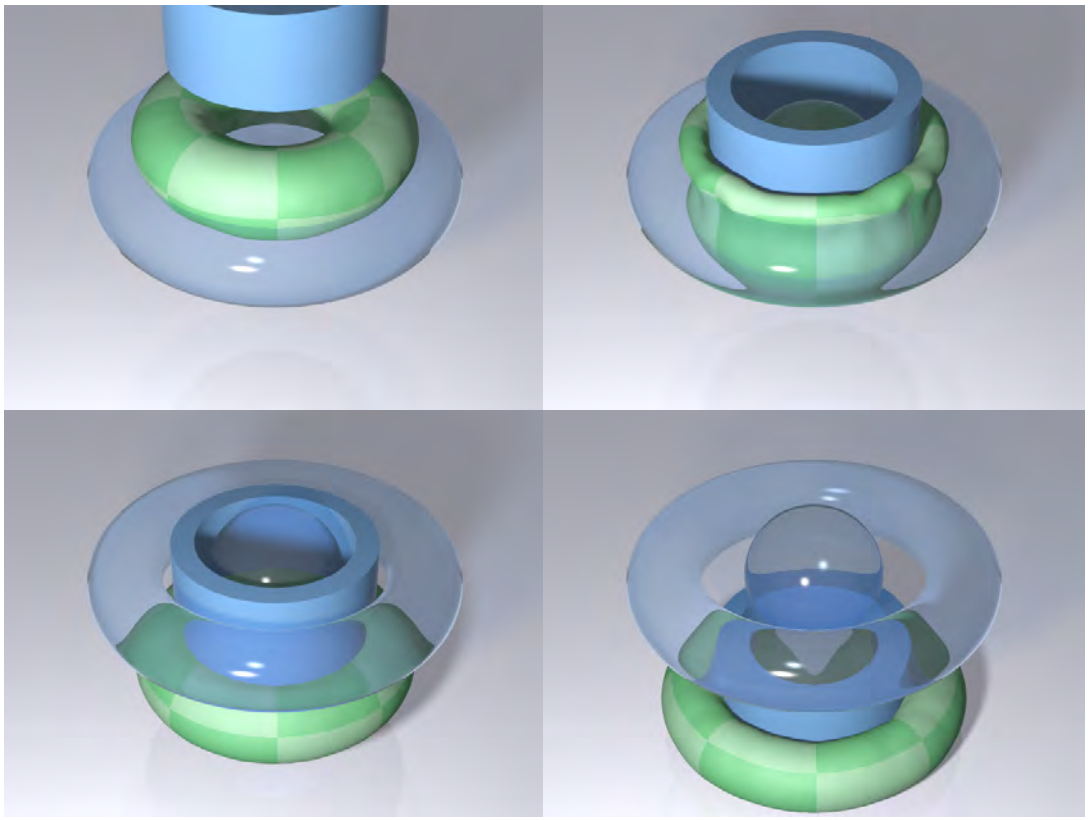

Figure A.10: *A torus is pushed through a hole (constraint collisions).*

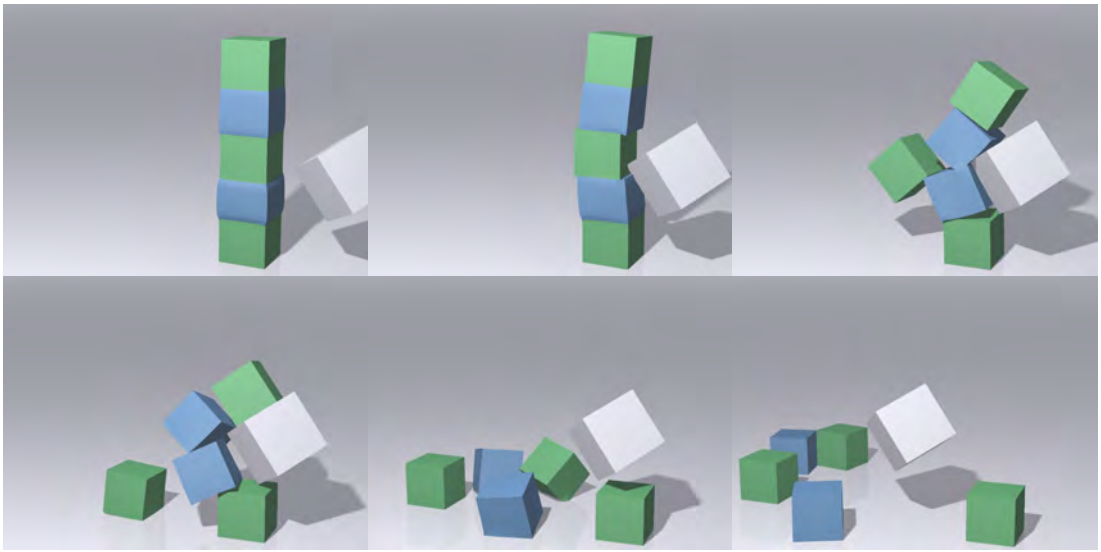

Figure A.11: *A stack of deformable boxes of varying stiffness is struck with a rigid kinematic cube (constraint collisions) with* $\Delta t = 1/24\,s$*. The green boxes are 10 times as stiff as the blue boxes.*

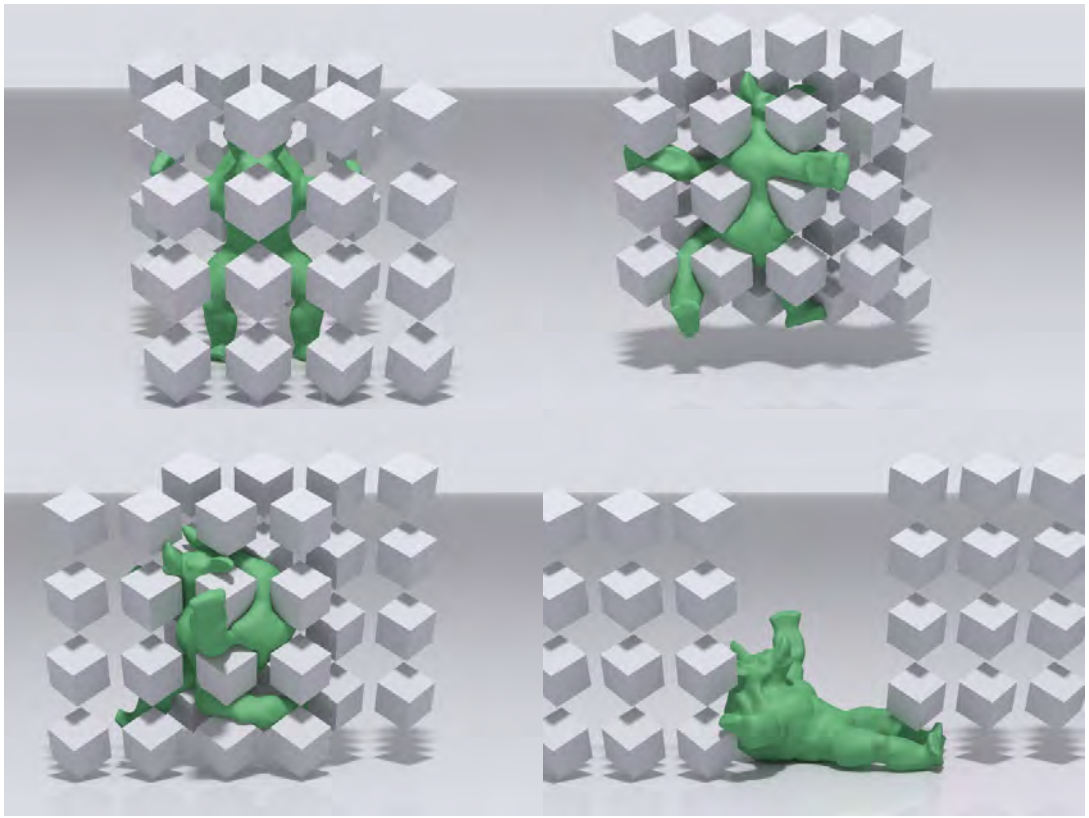

Figure A.12: *An armadillo is squeezed between 32 rigid cubes (constraint collisions) with $\Delta t = 1/24\,s$. When this torture test is run at 1, 2, 4 and 8 steps per frame the average runtime per frame is 46, 58, 88, and 117 seconds, respectively.*

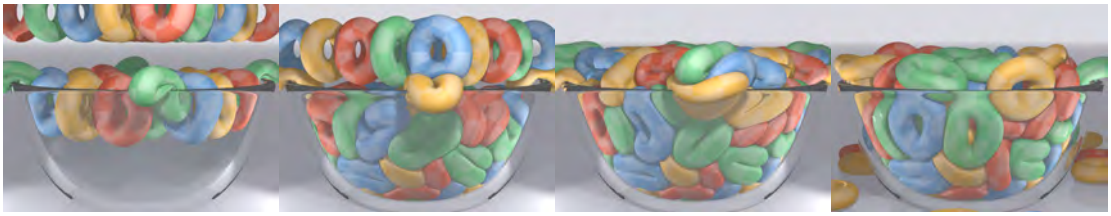

Figure A.13: *125 tori are dropped into a bowl at 5 time steps per frame, resulting in significant deformation and tough collisions.*

solver tests—high stress, terrible initial guess, tangled configurations, and the need to dissipate massive amounts of unwanted energy. Figure A.7 shows the recovery of a $65 \times 65 \times 65$ cube (824k dofs) from a randomized initial configuration for three different stiffnesses with $\Delta t = 1/24$ s. Figure A.8 repeats the tests with all points starting at the origin. The recovery times vary from about 3 s for the softest to a single time step for the stiffest. We were surprised to find that a single step of backward Euler could untangle a randomized cube, even at high resolution.

Figure A.9 is a classical torus drop demonstrating that our self collisions are effective at stopping collisions at the torus's hole. Figure A.10 uses constraints for all body collisions and demonstrates that our constraint collisions are effective with concave and convex constraint manifolds. Figure A.11 demonstrates our method with stiffer deformable bodies with sharp corners. Figure A.12 demonstrates our constraint collisions are effective for objects with sharp corners. Finally, Figure A.13 shows a more practical example, which uses all three types of collisions—self collisions, constraint collisions (with ground) and penalty collisions (against a bowl defined by a grid-based level set).

In Section 2.3, we demonstrate the advantages of using our optimization integrator by applying it to the MPM snow formulation from [Stomakhin et al., 2013].

A.8 Summary

We have demonstrated that backward Euler solved with Newton's method can be made more robust by recasting the resulting system of nonlinear equations as a nonlinear optimization problem so that robust optimization techniques can be employed. The resulting method is extremely robust to large time step sizes, high stress, and tangled configurations.

Runtimes and other performance-related information for all of our simulations are pro-

Figure	Ours	$\frac{\text{Steps}}{\text{frame}}$	$\frac{\text{Time}}{\text{frame}}$ (s)	# dofs	$\frac{\text{Solves}}{\text{step}}$
A.13	Y	5	200	984k	2.2
A.3 mid	Y	1	0.51	18.5k	2.8
A.3 rt	N	1/3	8.7/1.1	18.5k	15/0.7
A.2 top	Y	1	0.52	18.5k	2.9
A.2 bot	N	1/5	3.8/1.3	18.5k	6.6/0.6
A.4 top	Y	1	4.25	28.0k	8.1
A.4 bot	N	1/6	33/7.3	28.0k	26/0.8
A.9	Y	5	1.13	7.9k	2.1
A.7 top	Y	1	68.0^{+}	824k	12.3
A.7 lt	Y	1	1470^{+}	824k	236.8
A.7 rt	Y	1	667^{+}	824k	109.6
A.8 top	Y	1	43.1^{+}	824k	10.7
A.8 lt	Y	1	831^{+}	824k	155.9
A.8 rt	Y	1	444^{+}	824k	88.8
A.6 top	Y	1	0.42	14.0k	3.8
A.6 bot	N	1^{*}	1.13	14.0k	9.8
A.10	Y	1	0.45	7.9k	8.6
A.12	Y	1	46.1	73.8k	34.7
A.11	Y	1	17.1	138k	6.9

Table A.1: *Time step sizes and average running times for the examples. The last column shows the average number of linear solves per time step. Each of the Newton's method examples fails to converge at the frame rate. For a fairer comparison, timing information for all but the one marked * is shown at the frame rate and the stable time step size. The stress tests marked $^{+}$ spend the majority of their time on the first frame or two due to the difficult initial state.*

vided in Table A.1. All Lagrangian simulations were run single-threaded on a 3.1–3.5 GHz Xeon core, the MPM simulations were run with 10 threads for Figure 2.3 and 12 threads for Figure 2.2 and Figure 2.1. Our solver's performance is competitive with a standard Newton solver for those examples where both were run. In general, we take more Newton steps but spend less time on each, and the resulting runtime for typical examples is about the same for the two solvers, though our solver is faster for all of the difficult examples. Setting a large time step size can actually be slower than a smaller one, even with the same solver. For time integrators (like backward Euler) that have a significant amount of damping at large time steps, constitutive models are often tuned to take into account the numerical damping. If the integrator is forced to simulate a por-

tion of a simulation at a smaller time step, the dynamic behavior can change noticeably. Solving with constraints is about the same speed as using penalty collisions.

Note that Figure A.13 and Figure A.9 were run with smaller time steps sizes to avoid collision artifacts. This indicates that a self-collision scheme that is more tolerant of large time steps is required. The scheme does not have problems with collisions between different objects at the frame rate as long as they are not too thin. Continuous collision detection could perhaps be used. We leave both of these problems for future work.

The current method has a couple disadvantages compared with current techniques. It requires a potential energy to exist (which is how most constitutive models are defined anyway) and is sensitive to discontinuities in this energy. The method also occasionally fails to make progress due to valley shaped kinks in our collision processing. In practice, this only occurs when the system is already fairly close to a solution, since otherwise any energy kinks are overwhelmed by the strong gradients in the objective. From a practical perspective, this means this sort of breakdown can be dealt with by simply ignoring it. This does, however, prevent the method from being absolutely robust. We leave this weakness to be addressed in future work.

Our method was derived and implemented on top of a backward Euler integrator, which is known for being very stable but quite damped. The nonlinear system of equations for other A-stable integrators such as the trapezoid rule and BDF-2 can also be readily converted into minimization form and solved similarly. Being second-order schemes, their use would reduce damping at large time steps, though the trapezoid rule's oscillatory properties should be taken into account.

APPENDIX B

Derivatives for the Oldroyd-B Model

While the potential energy contains many elements and computing its second derivatives seems like a hopeless task, this is not the case. Breaking the potential energy into small pieces makes the implementation straightforward to implement and debug. In this appendix, we present pseudo-code that may be used to compute the potential energy $\Phi = \sum_p \Phi_p$ along with its derivatives, $\frac{\partial \Phi}{\partial \boldsymbol{x}_\mathbf{i}} = \sum_p \Phi_{p,\mathbf{i}}$ and $\frac{\partial^2 \Phi}{\partial \boldsymbol{x}_\mathbf{i} \partial \boldsymbol{x}_\mathbf{j}} = \sum_p \Phi_{p,\mathbf{ij}}$. The following computational steps may be used to compute the potential energy contribution of a particle Φ_p. Note that all the quantities computed below, except for the final result Φ_p, are intermediate quantities used to break the computation into many parts. Most of them have no particular physical significance, and most have no particular relationship to similarly named quantities elsewhere in this manuscript. The bold capitalized quantities are matrices, and the rest are scalars.

$$\mathbf{A}_p \leftarrow \sum_\mathbf{i} (\hat{\boldsymbol{x}}_\mathbf{i} - \boldsymbol{x}_\mathbf{i}^n)(\nabla w_{\mathbf{i}p}^n)^T \qquad \mathbf{B}_p \leftarrow \mathbf{A}_p \mathbf{b}_{OBp}^{E^n}$$

$$\mathbf{G}_p \leftarrow \mathbf{b}_{OBp}^{E^n} + \frac{\Delta t}{Wi}\left(\mathbf{I} - \mathbf{b}_{OBp}^{E^n}\right) \qquad \hat{\boldsymbol{F}}_p \leftarrow (\mathbf{I} + \mathbf{A}_p)\boldsymbol{F}_p^n$$

$$\mathbf{S}_p \leftarrow \mathbf{G}_p + \mathbf{B}_p + \mathbf{B}_p^T \qquad \boldsymbol{H}_p \leftarrow \hat{\boldsymbol{F}}_p^{-1}$$

$$J_p \leftarrow \det(\hat{\boldsymbol{F}}_p) \qquad a_p \leftarrow \frac{\lambda}{2}(J_p - 1)^2$$

$$q_p \leftarrow \frac{1}{2\Delta t^2}\left(\|\mathbf{A}_p\|_F^2 + \mathbf{A}_p^T : \mathbf{A}_p\right) \qquad b_p \leftarrow \mu \ln(J_p)$$

$$c_p \leftarrow \mu_P^N q_p \det(\boldsymbol{F}_p^n) \qquad g_p \leftarrow \mathrm{tr}(\mathbf{S}_p)$$

$$\boldsymbol{K}_p \leftarrow \mathbf{S}_p^{-1} \qquad h_p \leftarrow \det(\mathbf{S}_p)$$

$$k_p \leftarrow h_p^{-\frac{1}{d}} \qquad m_p \leftarrow J_p^{\frac{2}{d}}$$

$$n_p \leftarrow k_p g_p \qquad p_p \leftarrow \frac{\mu}{2} m_p n_p$$

$$\Phi_p \leftarrow V_p(p_p - b_p + a_p + c_p)$$

The next set of routines are for the first derivatives of the quantities above, with the final result being the potential energy derivative for a particle, $\Phi_{p,\mathbf{i}}$. Note that these routines use the quantities computed above. Intermediate quantities of the form $c_{p,\mathbf{i}}$ are related to the intermediates above by $c_{p,\mathbf{i}} = \frac{\partial c_p}{\partial \hat{\mathbf{x}}_\mathbf{i}}$, which allows for incremental testing. All quantities computed below are vectors.

$$\overline{b}_{p\mathbf{i}} \leftarrow \mathbf{b}_{OB_p}^{E^n} \nabla w_{\mathbf{i}p}^n \qquad \overline{f}_{p\mathbf{i}} \leftarrow (\boldsymbol{F}_p^n)^T \nabla w_{\mathbf{i}p}^n$$

$$\overline{h}_{p\mathbf{i}} \leftarrow \boldsymbol{H}_p^T \overline{f}_{p\mathbf{i}} \qquad \overline{k}_{p\mathbf{i}} \leftarrow \boldsymbol{K}_p^T \overline{b}_{p\mathbf{i}}$$

$$J_{p,\mathbf{i}} \leftarrow J_p \overline{h}_{p\mathbf{i}} \qquad a_{p,\mathbf{i}} \leftarrow \lambda(J_p - 1) J_{p,\mathbf{i}}$$

$$q_{p,\mathbf{i}} \leftarrow \frac{1}{\Delta t^2}(\mathbf{A}_p \nabla w_{\mathbf{i}p}^n + \mathbf{A}_p^T \nabla w_{\mathbf{i}p}^n) \qquad b_{p,\mathbf{i}} \leftarrow \mu \overline{h}_{p\mathbf{i}}$$

$$c_{p,\mathbf{i}} \leftarrow \mu_P^N \det(\boldsymbol{F}_p^n) q_{p,\mathbf{i}} \qquad g_{p,\mathbf{i}} \leftarrow 2\overline{b}_{p\mathbf{i}}$$

$$k_{p,\mathbf{i}} \leftarrow -\frac{2k_p}{d} \overline{k}_{p\mathbf{i}} \qquad m_{p,\mathbf{i}} \leftarrow \frac{2m_p}{d} \overline{h}_{p\mathbf{i}}$$

$$p_{p,\mathbf{i}} \leftarrow \frac{\mu}{2}(m_{p,\mathbf{i}} n_p + m_p n_{p,\mathbf{i}}) \qquad n_{p,\mathbf{i}} \leftarrow k_{p,\mathbf{i}} g_p + k_p g_{p,\mathbf{i}}$$

$$\Phi_{p,\mathbf{i}} \leftarrow V_p(p_{p,\mathbf{i}} - b_{p,\mathbf{i}} + a_{p,\mathbf{i}} + c_{p,\mathbf{i}})$$

The final set of routines are for second derivatives, with the final result being the potential energy Hessian for a particle, $\Phi_{p,\mathbf{ij}}$. Intermediate quantities of the form $c_{p,\mathbf{ij}}$ are related to the intermediates above by $c_{p,\mathbf{ij}} = \frac{\partial c_{p,\mathbf{i}}}{\partial \hat{\mathbf{x}}_\mathbf{j}}$. All quantities computed below are matrices.

$$J_{p,\mathbf{ij}} \leftarrow J_p \overline{h}_{p,\mathbf{i}} \overline{h}_{p,\mathbf{j}}^T - J_p \overline{h}_{p,\mathbf{j}} \overline{h}_{p,\mathbf{i}}^T$$

$$a_{p,\mathbf{ij}} \leftarrow \lambda J_{p,\mathbf{i}} J_{p,\mathbf{j}}^T + \lambda(J_p - 1) J_{p,\mathbf{ij}}$$

$$b_{p,\mathbf{ij}} \leftarrow -\mu \overline{h}_{p,\mathbf{j}} \overline{h}_{p,\mathbf{i}}^T$$

$$
\begin{aligned}
q_{p,\mathbf{ij}} &\leftarrow \frac{1}{\Delta t^2}((\nabla w_{\mathbf{ip}}^n)^T \nabla w_{,\mathbf{j}} \mathbf{I} + \nabla w_{,\mathbf{j}} (\nabla w_{\mathbf{ip}}^n)^T) \\
c_{p,\mathbf{ij}} &\leftarrow \mu_P^N \det(\boldsymbol{F}_p^n) q_{p,\mathbf{ij}} \\
k_{p,\mathbf{ij}} &\leftarrow \frac{4k_p}{d^2} \overline{k}_{p,\mathbf{i}} \overline{k}_{p,\mathbf{j}}^T + \frac{2k_p}{d} \overline{k}_{p,\mathbf{j}} \overline{k}_{p,\mathbf{i}}^T + \frac{2k_p}{d} \overline{b}_{p,\mathbf{i}}^T \overline{k}_{p,\mathbf{j}} \boldsymbol{K}_p \\
m_{p,\mathbf{ij}} &\leftarrow \frac{4m_p}{d^2} \overline{h}_{p,\mathbf{i}} \overline{h}_{p,\mathbf{j}}^T - \frac{2m_p}{d} \overline{h}_{p,\mathbf{j}} \overline{h}_{p,\mathbf{i}}^T \\
n_{p,\mathbf{ij}} &\leftarrow k_{p,\mathbf{ij}} g_p + k_{p,\mathbf{i}} g_{,\mathbf{j}}^T + g_{p,\mathbf{i}} k_{,\mathbf{j}}^T \\
p_{p,\mathbf{ij}} &\leftarrow \frac{\mu}{2}(m_{p,\mathbf{ij}} n_p + m_{p,\mathbf{i}} n_{,\mathbf{j}}^T + n_{p,\mathbf{i}} m_{,\mathbf{j}}^T + n_{p,\mathbf{ij}} m_p) \\
\Phi_{p,\mathbf{ij}} &\leftarrow V_p(p_{p,\mathbf{ij}} - b_{p,\mathbf{ij}} + a_{p,\mathbf{ij}} + c_{p,\mathbf{ij}})
\end{aligned}
$$

APPENDIX C

RPIC and APIC Proofs

This appendix provides detailed proofs for the properties of RPIC and APIC.

C.1 Preliminaries

When we consider conservation of angular momentum when transferring from the grid to particles at the end of a time step, we need to consider angular momentum to be defined over moved grid nodes and we use the notation $\tilde{\boldsymbol{x}}_i^{n+1} = \boldsymbol{x}_i + \Delta t \tilde{\boldsymbol{v}}_i^{n+1}$ to indicate this. To avoid confusion, rather than referring to unmoved grid nodes at the beginning of the time step as $\boldsymbol{x}_i$, we will use $\boldsymbol{x}_i^n$ to emphasize that they have not been dynamically updated yet, whereas the $\tilde{\boldsymbol{x}}_i^{n+1}$ have been updated.

We will also use a few properties of standard interpolating functions, namely.

$$\sum_i w_{ip} = 1$$

$$\sum_i w_{ip} \boldsymbol{x}_i^n = \boldsymbol{x}_p^n$$

$$\sum_i w_{ip} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) = \mathbf{0}$$

C.2 Piecewise rigid

The transfer from particles to the grid is given by

$$
\begin{aligned}
m_i^n &= \sum_p w_{ip}^n m_p \\
\boldsymbol{K}_p^n &= \sum_j w_{jp}^n m_p (\boldsymbol{x}_j^n - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_j^n - \boldsymbol{x}_p^n)^{*T} \\
m_i^n \boldsymbol{v}_i^n &= \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + ((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n))
\end{aligned}
$$

with the transfer to particles given by

$$
\begin{aligned}
\boldsymbol{v}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \\
\mathbf{L}_p^{n+1} &= \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p \tilde{\boldsymbol{v}}_i^{n+1}.
\end{aligned}
$$

C.2.1 Preservation of rigid motion

Let $\Delta t = 0$ and consider the the process of transferring velocity ($\tilde{\boldsymbol{v}}_i^{n+1}$) information to particles ($\boldsymbol{v}_p^{n+1}$, $\mathbf{B}_p^{n+1}$) and then back to the grid ($\boldsymbol{v}_i^{n+1}$). Since $\Delta t = 0$, we have $w_{ip}^n = w_{ip}^{n+1}$ and $\boldsymbol{x}_p^n = \boldsymbol{x}_p^{n+1}$, so that $\boldsymbol{K}_p^n = \boldsymbol{K}_p^{n+1}$ and $m_i^n = m_i^{n+1}$. If the velocities before the transfer represent rigid motion, then $\tilde{\boldsymbol{v}}_i^{n+1} = \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n$, where $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are vectors.

$$
\begin{aligned}
\tilde{\boldsymbol{v}}_i^{n+1} &= \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n \\
\boldsymbol{v}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i w_{ip}^n (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n) \\
&= \sum_i w_{ip}^n \boldsymbol{v} + \sum_i w_{ip}^n \boldsymbol{\omega} \times \boldsymbol{x}_i^n
\end{aligned}
$$

$$
\begin{aligned}
&= \boldsymbol{v} \sum_i w_{ip}^n + \boldsymbol{\omega} \times \sum_i w_{ip}^n \boldsymbol{x}_i^n \\
&= \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_p^n \\
\mathbf{L}_p^{n+1} &= \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n) \\
&= \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right) \times m_p \boldsymbol{v} + \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p (\boldsymbol{\omega} \times \boldsymbol{x}_i^n) \\
&= \sum_i m_p w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_i^n)^{*T} \boldsymbol{\omega} \\
&= \sum_i m_p w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^{*T} \boldsymbol{\omega} + \sum_i m_p w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_p^n)^{*T} \boldsymbol{\omega} \\
&= \boldsymbol{K}_p^n \boldsymbol{\omega} + m_p \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right)^* (\boldsymbol{x}_p^n)^{*T} \boldsymbol{\omega} \\
&= \boldsymbol{K}_p^n \boldsymbol{\omega}
\end{aligned}
$$

$$
\begin{aligned}
m_i^{n+1} \boldsymbol{v}_i^{n+1} &= \sum_p w_{ip}^{n+1} m_p (\boldsymbol{v}_p^{n+1} + ((\boldsymbol{K}_p^{n+1})^{-1} \mathbf{L}_p^{n+1}) \times (\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_p^{n+1})) \\
m_i^n \boldsymbol{v}_i^{n+1} &= \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^{n+1} + ((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^{n+1}) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_p w_{ip}^n m_p (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_p^n + ((\boldsymbol{K}_p^n)^{-1} \boldsymbol{K}_p^n \boldsymbol{\omega}) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_p w_{ip}^n m_p (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_p^n + \boldsymbol{\omega} \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \left(\sum_p w_{ip}^n m_p \right) (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n) \\
&= m_i^n (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n) \\
\boldsymbol{v}_i^{n+1} &= \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{x}_i^n \\
&= \tilde{\boldsymbol{v}}_i^{n+1}
\end{aligned}
$$

C.2.2 Conservation of momentum

Particle to grid The angular momentum on the grid after transferring from particles is

$$
\begin{aligned}
\mathbf{p}_{tot}^{G,n} &= \sum_i m_i^n \boldsymbol{v}_i^n \\
&= \sum_i \left(\sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + ((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \right) \\
&= \sum_{i,p} w_{ip}^n m_p \boldsymbol{v}_p^n + \sum_{i,p} w_{ip}^n m_p (((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_p \left(\sum_i w_{ip}^n \right) m_p \boldsymbol{v}_p^n + \sum_p m_p ((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right) \\
&= \sum_p m_p \boldsymbol{v}_p^n \\
&= \mathbf{p}_{tot}^{P,n}
\end{aligned}
$$

Grid to particle The angular momentum on the particles after transferring from the grid is

$$
\begin{aligned}
\mathbf{p}_{tot}^{P,n+1} &= \sum_p m_p \boldsymbol{v}_p^{n+1} \\
&= \sum_p m_p \left(\sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \right) \\
&= \sum_i \left(\sum_p w_{ip}^n m_p \right) \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i m_i^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \mathbf{p}_{tot}^{G,n+1}
\end{aligned}
$$

C.2.3 Conservation of angular momentum

Particle to grid The angular momentum on the grid after transferring from particles is

$$
\begin{aligned}
\mathbf{L}_{tot}^{G,n} &= \sum_i \boldsymbol{x}_i^n \times m_i^n \boldsymbol{v}_i^n \\
&= \sum_i \boldsymbol{x}_i^n \times \left(\sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + ((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \right) \\
&= \sum_{i,p} \boldsymbol{x}_i^n \times w_{ip}^n m_p \boldsymbol{v}_p^n + \sum_{i,p} \boldsymbol{x}_i^n \times w_{ip}^n m_p (((\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n) \times (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_p \left(\sum_i w_{ip}^n \boldsymbol{x}_i^n \right) \times m_p \boldsymbol{v}_p^n + \sum_{i,p} \boldsymbol{x}_i^n \times w_{ip}^n m_p (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^{*T} (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&= \sum_p \boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \sum_{i,p} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times w_{ip}^n m_p (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^{*T} (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&\quad + \sum_{i,p} \boldsymbol{x}_p^n \times w_{ip}^n m_p (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^{*T} (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&= \sum_p \boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \sum_p \left(\sum_i m_p w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^* (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^{*T} \right) (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&\quad + \sum_p \boldsymbol{x}_p^n \times m_p \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right)^{*T} (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&= \sum_p \boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \sum_p \boldsymbol{K}_p^n (\boldsymbol{K}_p^n)^{-1} \mathbf{L}_p^n \\
&= \sum_p (\boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \mathbf{L}_p^n) \\
&= \mathbf{L}_{tot}^{P,n}
\end{aligned}
$$

Grid to particle The angular momentum on the particles after transferring from the grid is

$$
\mathbf{L}_{tot}^{P,n+1} = \sum_p (\boldsymbol{x}_p^{n+1} \times m_p \boldsymbol{v}_p^{n+1} + \mathbf{L}_p^{n+1})
$$

$$
\begin{aligned}
&= \sum_p \left(\boldsymbol{x}_p^{n+1} \times m_p \left(\sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \right) + \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \right) \right) \\
&= \sum_{i,p} \left(\boldsymbol{x}_p^{n+1} \times m_p w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} + w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \right) \\
&= \sum_{i,p} (\boldsymbol{x}_p^{n+1} - \boldsymbol{x}_p^n) \times m_p w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} + \sum_{i,p} w_{ip}^n \boldsymbol{x}_i^n \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \Delta t \sum_p \boldsymbol{v}_p^{n+1} \times m_p \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} + \sum_{i,p} w_{ip}^n (\tilde{\boldsymbol{x}}_i^{n+1} - \Delta t \tilde{\boldsymbol{v}}_i^{n+1}) \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \Delta t \sum_{i,p} \boldsymbol{v}_p^{n+1} \times m_p \boldsymbol{v}_p^{n+1} + \sum_{i,p} w_{ip}^n \tilde{\boldsymbol{x}}_i^{n+1} \times m_p \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i \left(\sum_p w_{ip}^n m_p \right) \tilde{\boldsymbol{x}}_i^{n+1} \times \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i \tilde{\boldsymbol{x}}_i^{n+1} \times m_i^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \mathbf{L}_{tot}^{G,n+1}
\end{aligned}
$$

C.3 Affine

The transfer from particles to the grid is given by

$$
\begin{aligned}
m_i^n &= \sum_p w_{ip}^n m_p \\
\mathbf{D}_p^n &= \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T = \sum_i w_{ip}^n \boldsymbol{x}_i^n (\boldsymbol{x}_i^n)^T - \boldsymbol{x}_p^n (\boldsymbol{x}_p^n)^T \\
m_i^n \boldsymbol{v}_i^n &= \sum_p w_{ip}^n m_p (\boldsymbol{v}_p^n + \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n))
\end{aligned}
$$

with the transfer to particles given by

$$
\begin{aligned}
\boldsymbol{v}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \\
\mathbf{B}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T.
\end{aligned}
$$

C.3.1 Preservation of affine velocity fields

Let $\Delta t = 0$ and consider the the process of transferring velocity ($\tilde{\boldsymbol{v}}_i^{n+1}$) information to particles ($\boldsymbol{v}_p^{n+1}$, $\mathbf{B}_p^{n+1}$) and then back to the grid ($\boldsymbol{v}_i^{n+1}$). Since $\Delta t = 0$, we have $w_{ip}^n = w_{ip}^{n+1}$ and $\boldsymbol{x}_p^n = \boldsymbol{x}_p^{n+1}$, so that $\mathbf{D}_p^n = \mathbf{D}_p^{n+1}$ and $m_i^n = m_i^{n+1}$. If the velocities before the transfer represent an affine velocity field, then $\tilde{\boldsymbol{v}}_i^{n+1} = \boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^n$, where $\boldsymbol{v}$ is a vector and $\mathbf{C}$ is a matrix.

$$
\begin{aligned}
\tilde{\boldsymbol{v}}_i^{n+1} &= \boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^n \\
\boldsymbol{v}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i w_{ip}^n (\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^n) \\
&= \sum_i w_{ip}^n \boldsymbol{v} + \sum_i w_{ip}^n \mathbf{C}\boldsymbol{x}_i^n \\
&= \boldsymbol{v} \sum_i w_{ip}^n + \mathbf{C} \sum_i w_{ip}^n \boldsymbol{x}_i^n \\
&= \boldsymbol{v} + \mathbf{C}\boldsymbol{x}_p^n \\
\mathbf{B}_p^{n+1} &= \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T \\
&= \sum_i w_{ip}^n (\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^n)(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T \\
&= \sum_i w_{ip}^n \boldsymbol{v}(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T + \sum_i w_{ip}^n \mathbf{C}\boldsymbol{x}_i^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T \\
&= \boldsymbol{v} \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right)^T + \mathbf{C} \sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T + \sum_i w_{ip}^n \mathbf{C}\boldsymbol{x}_p^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)^T \\
&= \mathbf{C}\mathbf{D}_p^n + \mathbf{C}\boldsymbol{x}_p^n \left(\sum_i w_{ip}^n (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \right)^T \\
&= \mathbf{C}\mathbf{D}_p^n
\end{aligned}
$$

$$
\begin{aligned}
m_i^{n+1}\boldsymbol{v}_i^{n+1} &= \sum_p w_{ip}^{n+1} m_p (\boldsymbol{v}_p^{n+1} + \mathbf{B}_p^{n+1}(\mathbf{D}_p^{n+1})^{-1}(\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_p^{n+1})) \\
m_i^{n}\boldsymbol{v}_i^{n+1} &= \sum_p w_{ip}^{n} m_p (\boldsymbol{v}_p^{n+1} + \mathbf{B}_p^{n+1}(\mathbf{D}_p^{n})^{-1}(\boldsymbol{x}_i^{n} - \boldsymbol{x}_p^{n})) \\
&= \sum_p w_{ip}^{n} m_p (\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_p^{n} + \mathbf{C}\mathbf{D}_p^{n}(\mathbf{D}_p^{n})^{-1}(\boldsymbol{x}_i^{n} - \boldsymbol{x}_p^{n})) \\
&= \sum_p w_{ip}^{n} m_p (\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_p^{n} + \mathbf{C}(\boldsymbol{x}_i^{n} - \boldsymbol{x}_p^{n})) \\
&= \left(\sum_p w_{ip}^{n} m_p\right)(\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^{n}) \\
&= m_i^{n}(\boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^{n}) \\
\boldsymbol{v}_i^{n+1} &= \boldsymbol{v} + \mathbf{C}\boldsymbol{x}_i^{n} \\
&= \tilde{\boldsymbol{v}}_i^{n+1}
\end{aligned}
$$

C.3.2 Conservation of momentum

Particle to grid

$$
\begin{aligned}
\mathbf{p}_{tot}^{G,n} &= \sum_i m_i^n \boldsymbol{v}_i^n \\
&= \sum_p \sum_i m_p w_{ip}^n (\boldsymbol{v}_p^n + \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1}(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_{p,i} m_p w_{ip}^n \boldsymbol{v}_p^n + \sum_{p,i} m_p w_{ip}^n \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1}(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n) \\
&= \sum_p m_p \left(\sum_i w_{ip}^n\right)\boldsymbol{v}_p^n + \sum_p m_p \mathbf{B}_p^n(\mathbf{D}_p^n)^{-1}\left(\sum_i w_{ip}^n(\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)\right) \\
&= \sum_p m_p \boldsymbol{v}_p^n \\
&= \mathbf{p}_{tot}^{P,n}
\end{aligned}
$$

Grid to particle

$$\begin{aligned}
\mathbf{p}_{tot}^{P,n+1} &= \sum_p m_p \boldsymbol{v}_p^{n+1} \\
&= \sum_p m_p \sum_i w_{ip}^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i \left(\sum_p m_p w_{ip}^n \right) \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \sum_i m_i^n \tilde{\boldsymbol{v}}_i^{n+1} \\
&= \mathbf{p}_{tot}^{G,n+1}
\end{aligned}$$

C.3.3 Conservation of angular momentum

Particle to grid

$$\begin{aligned}
\mathbf{L}_{tot}^{G,n} &= \sum_i \boldsymbol{x}_i^n \times m_i^n \boldsymbol{v}_i^n \\
&= \sum_p \sum_i \boldsymbol{x}_i^n \times m_p w_{ip}^n (\boldsymbol{v}_p^n + \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} (\boldsymbol{x}_i^n - \boldsymbol{x}_p^n)) \\
&= \sum_p \sum_i \boldsymbol{x}_i^n \times m_p w_{ip}^n \boldsymbol{v}_p^n + \sum_p \sum_i \boldsymbol{x}_i^n \times m_p w_{ip}^n \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} \boldsymbol{x}_i^n \\
&\quad - \sum_p \sum_i \boldsymbol{x}_i^n \times m_p w_{ip}^n \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} \boldsymbol{x}_p^n \\
&= \sum_p \boldsymbol{x}_p^n \times m_p \boldsymbol{v}_p^n + \sum_p \sum_i \boldsymbol{x}_i^n \times m_p w_{ip}^n \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} \boldsymbol{x}_i^n \\
&\quad - \sum_p \boldsymbol{x}_p^n \times m_p \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1} \boldsymbol{x}_p^n \\
\left(\mathbf{L}_{tot}^{G,n} \right)_\alpha &= \sum_{p,\beta,\gamma} x_{p\beta}^n e_{\beta\gamma\alpha} m_p v_{p\gamma}^n + \sum_{p,i,\beta,\gamma,\tau,\sigma} x_{i\beta}^n e_{\beta\gamma\alpha} m_p w_{ip}^n B_{p\gamma\tau}^n (D^{-1})_{p\tau\sigma}^n x_{i\sigma}^n \\
&\quad - \sum_{p,\beta,\gamma,\tau,\sigma} x_{p\beta}^n e_{\beta\gamma\alpha} m_p B_{p\gamma\tau}^n (D^{-1})_{p\tau\sigma}^n x_{p\sigma}^n
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{p,\beta,\gamma} x^n_{p\beta} e_{\beta\gamma\alpha} m_p v^n_{p\gamma} \\
&\quad + \sum_{p,\beta,\gamma,\tau,\sigma} e_{\beta\gamma\alpha} m_p B^n_{p\gamma\tau} (D^{-1})^n_{p\tau\sigma} \left(\sum_i x^n_{i\beta} w^n_{ip} x^n_{i\sigma} - x^n_{p\beta} x^n_{p\sigma} \right) \\
&= \sum_{p,\beta,\gamma} x^n_{p\beta} e_{\beta\gamma\alpha} m_p v^n_{p\gamma} + \sum_{p,\beta,\gamma,\tau,\sigma} e_{\beta\gamma\alpha} m_p B^n_{p\gamma\tau} (D^{-1})^n_{p\tau\sigma} D^n_{p\sigma\beta} \\
&= \sum_{p,\beta,\gamma} x^n_{p\beta} e_{\beta\gamma\alpha} m_p v^n_{p\gamma} + \sum_{p,\beta,\gamma} e_{\beta\gamma\alpha} m_p B^n_{p\gamma\beta} \\
\mathbf{L}^{G,n}_{tot} &= \sum_p \boldsymbol{x}^n_p \times m_p \boldsymbol{v}^n_p + \sum_p m_p (\mathbf{B}^n_p)^T : \boldsymbol{\epsilon} \\
&= \mathbf{L}^{P,n}_{tot}
\end{aligned}
$$

Grid to particle

$$
\begin{aligned}
\mathbf{L}^{P,n+1}_{tot} &= \sum_p \boldsymbol{x}^{n+1}_p \times m_p \boldsymbol{v}^{n+1}_p + \sum_p m_p (\mathbf{B}^{n+1}_p)^T : \boldsymbol{\epsilon} \\
\left(\mathbf{L}^{P,n+1}_{tot} \right)_\alpha &= \sum_{p,\beta,\gamma} e_{\beta\gamma\alpha} x^{n+1}_{p\beta} m_p v^{n+1}_{p\gamma} + \sum_{p,\beta,\gamma} e_{\beta\gamma\alpha} m_p B^{n+1}_{p\gamma\beta} \\
&= \sum_{p,\beta,\gamma} e_{\beta\gamma\alpha} x^{n+1}_{p\beta} m_p \sum_i w^n_{ip} \tilde{v}^{n+1}_{i\gamma} + \sum_{p,\beta,\gamma} e_{\beta\gamma\alpha} m_p \sum_i w^n_{ip} \tilde{v}^{n+1}_{i\gamma} (x^n_{i\beta} - x^n_{p\beta}) \\
\mathbf{L}^{P,n+1}_{tot} &= \sum_{p,i} \boldsymbol{x}^{n+1}_p \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i + \sum_{p,i} (\boldsymbol{x}^n_i - \boldsymbol{x}^n_p) \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i \\
&= \sum_{p,i} (\boldsymbol{x}^{n+1}_p - \boldsymbol{x}^n_p) \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i + \sum_{p,i} \boldsymbol{x}^n_i \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i \\
&= \Delta t \sum_p \boldsymbol{v}^{n+1}_p \times m_p \sum_i w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i + \sum_{p,i} (\tilde{\boldsymbol{x}}^{n+1}_i - \Delta t \tilde{\boldsymbol{v}}^{n+1}_i) \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i \\
&= \Delta t \sum_{p,i} \boldsymbol{v}^{n+1}_p \times m_p \boldsymbol{v}^{n+1}_p + \sum_{p,i} \tilde{\boldsymbol{x}}^{n+1}_i \times m_p w^n_{ip} \tilde{\boldsymbol{v}}^{n+1}_i \\
&= \sum_i \tilde{\boldsymbol{x}}^{n+1}_i \times m^n_i \tilde{\boldsymbol{v}}^{n+1}_i \\
&= \mathbf{L}^{G,n+1}_{tot}
\end{aligned}
$$

APPENDIX D

Derivatives for Deviatoric Elasticity

For our constitutive model we use $\hat{\Psi}_\mu(\boldsymbol{F}) = \Psi_\mu(J^{-\frac{1}{d}}\boldsymbol{F})$, where plasticity does not matter and is ignored for the purposes of computing these derivatives. For convenience, let $a = -\frac{1}{d}$, and the μ subscripts are ignored. Then, $\hat{\Psi}(\boldsymbol{F}) = \Psi(J^a\boldsymbol{F})$. We begin by computing $\hat{\Psi}_\mu(\boldsymbol{F})$. We will use index notation for precision during the derivations. Differentiation by the matrix $\boldsymbol{F}_{ij}$ is indicated by enclosing the index pair in parenthesis after a comma, as in $J_{,(ij)}$. Let $\boldsymbol{H} = \boldsymbol{F}^{-T}$. We begin with some preliminary derivatives for J and $\boldsymbol{H}$.

$$
\begin{aligned}
H_{ji}F_{jk} &= \delta_{ik} \\
J_{,(ij)} &= JH_{ij} \\
(J^a)_{,(ij)} &= aJ^{a-1}J_{,(ij)} \\
&= aJ^{a-1}JH_{ij} \\
&= aJ^aH_{ij} \\
(H_{ji}F_{jk})_{,(rs)} &= 0 \\
H_{ji,(rs)}F_{jk} + H_{ji}F_{jk,(rs)} &= 0 \\
H_{ji,(rs)}F_{jk} &= -H_{ji}F_{jk,(rs)} \\
H_{ji,(rs)}\delta_{jm} &= -H_{ji}\delta_{jr}\delta_{ks}H_{mk} \\
H_{ji,(rs)} &= -H_{ri}H_{js}
\end{aligned}
$$

The derivatives of the quantity $J^a\boldsymbol{F}$ will occur frequently, so we begin by naming them and evaluating them.

$$
\begin{aligned}
\boldsymbol{K}_{kmij} &= (J^a F_{km})_{,(ij)} \\
&= J^a F_{km,(ij)} + (J^a)_{,(ij)} F_{km} \\
&= J^a \delta_{ik}\delta_{jm} + a J^a F_{km} H_{ij} \\
\boldsymbol{K}_{kmij} Z_{ij} &= J^a \delta_{ik}\delta_{jm} Z_{ij} + a J^a F_{km} H_{ij} Z_{ij} \\
\boldsymbol{K}_{kmij} Z_{ij} &= J^a Z_{km} + a J^a F_{km} H_{ij} Z_{ij} \\
\boldsymbol{K} : \boldsymbol{Z} &= J^a (\boldsymbol{Z} + a(\boldsymbol{H} : \boldsymbol{Z})\boldsymbol{F}) \\
Z_{km}\boldsymbol{K}_{kmij} &= J^a \delta_{ik}\delta_{jm} Z_{km} + a J^a F_{km} H_{ij} Z_{km} \\
Z_{km}\boldsymbol{K}_{kmij} &= J^a Z_{ij} + a J^a F_{km} Z_{km} H_{ij} \\
\boldsymbol{Z} : \boldsymbol{K} &= J^a (\boldsymbol{Z} + a(\boldsymbol{F} : \boldsymbol{Z})\boldsymbol{H}) \\
\boldsymbol{K}_{kmij,(rs)} &= (J^a(\delta_{ik}\delta_{jm} + a F_{km} H_{ij}))_{,(rs)} \\
\boldsymbol{K}_{kmij,(rs)} &= (J^a)_{,(rs)}(\delta_{ik}\delta_{jm} + a F_{km} H_{ij}) + J^a(\delta_{ik}\delta_{jm} + a F_{km} H_{ij})_{,(rs)} \\
\boldsymbol{K}_{kmij,(rs)} &= a J^a H_{rs}(\delta_{ik}\delta_{jm} + a F_{km} H_{ij}) + a J^a (F_{km,(rs)} H_{ij} + F_{km} H_{ij,(rs)}) \\
\boldsymbol{K}_{kmij,(rs)} &= a \boldsymbol{K}_{kmij} H_{rs} + a J^a (\delta_{kr}\delta_{ms} H_{ij} - F_{km} H_{rj} H_{is})
\end{aligned}
$$

With the operator $\boldsymbol{K}$, we can express the relationship between $\hat{\mathbf{A}} = \frac{\partial \hat{\Psi}}{\partial \boldsymbol{F}}(\boldsymbol{F})$ and $\mathbf{A} = \frac{\partial \Psi}{\partial \boldsymbol{F}}(J^a \boldsymbol{F})$.

$$
\begin{aligned}
\hat{\Psi}(F_{ij}) &= \Psi(J^a F_{ij}) \\
\hat{\Psi}_{,(ij)} &= \Psi_{,(km)}(J^a F_{km})_{,(ij)} \\
&= \Psi_{,(km)} \boldsymbol{K}_{kmij} \\
\hat{\mathbf{A}} &= \mathbf{A} : \boldsymbol{K}
\end{aligned}
$$

Finally, we relate $\mathbf{C} = \frac{\partial^2 \Psi}{\partial \boldsymbol{F} \partial \boldsymbol{F}}(J^a \boldsymbol{F})$ to $\hat{\mathbf{C}} = \frac{\partial^2 \hat{\Psi}}{\partial \boldsymbol{F} \partial \boldsymbol{F}}(\boldsymbol{F})$.

$$
\begin{aligned}
\hat{\Psi}_{,(ij)(rs)} &= \left(\Psi_{,(km)} \boldsymbol{K}_{kmij}\right)_{,(rs)} \\
\hat{\Psi}_{,(ij)(rs)} &= \Psi_{,(km)(tu)} \boldsymbol{K}_{turs} \boldsymbol{K}_{kmij} + \Psi_{,(km)} \boldsymbol{K}_{kmij,(rs)} \\
\hat{\Psi}_{,(ij)(rs)} &= \Psi_{,(km)(tu)} \boldsymbol{K}_{turs} \boldsymbol{K}_{kmij} \\
&\quad + a \Psi_{,(km)} \boldsymbol{K}_{kmij} H_{rs} + a J^a \Psi_{,(rs)} H_{ij} - a J^a \Psi_{,(km)} F_{km} H_{rj} H_{is} \\
\hat{\Psi}_{,(ij)(rs)} Z_{rs} &= \Psi_{,(km)(tu)} \boldsymbol{K}_{turs} \boldsymbol{K}_{kmij} Z_{rs} + a \Psi_{,(km)} \boldsymbol{K}_{kmij} H_{rs} Z_{rs} \\
&\quad + a J^a \Psi_{,(rs)} H_{ij} Z_{rs} - a J^a \Psi_{,(km)} F_{km} H_{rj} H_{is} Z_{rs} \\
\hat{\mathbf{C}} : \boldsymbol{Z} &= (\mathbf{C} : (\boldsymbol{K} : \boldsymbol{Z})) : \boldsymbol{K} + a (\boldsymbol{H} : \boldsymbol{Z}) \mathbf{A} : \boldsymbol{K} \\
&\quad + a J^a (\mathbf{A} : \boldsymbol{Z}) \boldsymbol{H} - a J^a (\mathbf{A} : \boldsymbol{F}) \boldsymbol{H} \boldsymbol{Z}^T \boldsymbol{H}
\end{aligned}
$$

BIBLIOGRAPHY

Ando, R., Thurey, N., and Tsuruno, R. (2012). Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans Vis Comp Graph*, 18(8):1202–1214. 40

Ando, R., Thurey, N., and Wojtan, C. (2013). Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans Graph*, 32(4):103:1–103:10. 40

Ando, R. and Tsuruno, R. (2011). A particle-based method for preserving fluid sheets. In *Proc ACM SIGGRAPH/Eurographics Symp Comp Anim*, SCA '11, pages 7–16. 40

Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. In *Proc. SIGGRAPH*, pages 43–54. 94

Bargteil, A., Wojtan, C., Hodgins, J., and Turk, G. (2007). A finite element method for animating large viscoplastic flow. *ACM Trans. Graph.*, 26(3). 20, 24, 50, 59, 72, 89

Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. *ACM Trans Graph*, 26(3). 37

Batty, C. and Bridson, R. (2008a). Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proc 2008 ACM/Eurographics Symp Comp Anim*, pages 219–228. 24, 37

Batty, C. and Bridson, R. (2008b). Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proc 2008 ACM/Eurographics Symp Comp Anim*, pages 219–228. 89

Batty, C. and Houston, B. (2011). A simple finite volume method for adaptive viscous liquids. In *Proc 2011 ACM SIGGRAPH/Eurograp Symp Comp Anim*, pages 111–118. 24

Batty, C., Uribe, A., Audoly, B., and Grinspun, E. (2012). Discrete viscous sheets. 31(4):113:1–113:7. 24

Becker, M., Ihmsen, M., and Teschner, M. (2009). Corotated sph for deformable solids. In *Eurographics Conf. Nat. Phen.*, pages 27–34. 25, 58, 59

Bonet, J. and Wood, R. (1997). *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press. 6, 26, 64, 67, 70, 72

Boyd, L. and Bridson, R. (2012). Multiflip for energetic two-phase fluid simulation. *ACM Trans Graph*, 31(2):16:1–16:12. 37

Brackbill, J. (1988). The ringing instability in particle-in-cell calculations of low-speed flow. *J Comp Phys*, 75(2):469–492. 39

Brackbill, J., Kothe, D., and Ruppel, H. (1988). Flip: A low-dissipation, pic method for fluid flow. *Comp Phys Comm*, 48:25–38. 38

Brackbill, J. and Ruppel, H. (1986). Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J Comp Phys*, 65:314–343. 38

Bridson, R. (2008). *Fluid simulation for computer graphics*. Taylor & Francis. 47

Bridson, R., Fedkiw, R., and Anderson, J. (2002). Robust treatment of collisions, contact and friction for cloth animation. In *ACM Trans. Graph. (ToG)*, volume 21, pages 594–603. ACM. 94

Bridson, R., Marino, S., and Fedkiw, R. (2003). Simulation of clothing with folds and wrinkles. In *Proc. Symp. Comp. Anim.*, pages 28–36. 94

Carlson, M., Mucha, P., Horn, R. V., and Turk, G. (2002a). Melting and flowing. In *ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 167–174. 24

Carlson, M., Mucha, P., and Turk, G. (2004). Rigid fluid: animating the interplay between rigid bodies and fluid. In *ACM Trans. on Graph.*, volume 23, pages 377–384. 61

Carlson, M., Mucha, P. J., Van Horn, III, R. B., and Turk, G. (2002b). Melting and flowing. In *ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 167–174. 60, 65, 89

Chang, Y., Bao, K., Liu, Y., Zhu, J., and Wu, E. (2009). A particle-based method for viscoelastic fluids animation. In *ACM Symp. Virt. Real. Soft. Tech.*, pages 111–117. 25, 58, 60, 65

Chao, I., Pinkall, U., Sanan, P., and Schroeder, P. (2010). A simple geometric model for elastic deformations. *ACM Trans. Graph.*, 29:38:1–38:6. 108

Chentanez, N., Goktekin, T. G., Feldman, B. E., and O'Brien, J. F. (2006). Simultaneous coupling of fluids and deformable bodies. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 83–89. 61

Chentanez, N. and Muller, M. (2010). Real-time simulation of large bodies of water with small scale details. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '10, pages 197–206. 37

Chentanez, N. and Muller, M. (2011). Real-time eulerian water simulation using a restricted tall cell grid. *ACM Trans Graph*, 30(4):82:1–82:10. 37

Chentanez, N. and Muller, M. (2014). Coupling 3d eulerian, height field and particle methods for the simulation of large scale liquid phenomena. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '14. 37

Choi, K.-J. and Ko, H.-S. (2005). Stable but responsive cloth. In *ACM SIGGRAPH 2005 Courses*, page 1. ACM. 94

Choi, S.-C. T. (2006). *Iterative Methods for Singular Linear Equations and Least-Squares Problems*. PhD thesis, ICME, Stanford University, CA. 78

Chorin, A. (1968). Numerical solution of the Navier-Stokes Equations. *Math. Comp.*, 22:745–762. 58, 70

Clausen, P., Wicke, M., Shewchuk, J. R., and O'brien, J. F. (2013). Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph.*, 32(2):17:1–17:15. 59, 60

Cornelis, J., Ihmsen, M., Peer, A., and Teschner, M. (2014). Iisph-flip for incompressible fluids. *Comp Graph Forum*, 33(2):255–262. 37

Dagenais, F., Gagnon, J., and Paquette, E. (2012). A prediction-correction approach for stable sph fluid simulation from liquid to rigid. In *Proc. of Comp. Graph. Intl.* 60, 65

Desbrun, M. and Gascuel, M. (1996). Smoothed particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop Comp. Anim. Sim.*, pages 61–76. 24, 58

Eberhardt, B., Etzmus, O., and Hauth, M. (2000). *Implicit-explicit schemes for fast animation with particle systems*. Springer. 94

Edwards, E. and Bridson, R. (2012). A high-order accurate particle-in-cell method. *Int J Numer Meth Eng*, 90:1073–1088. 37, 40

Edwards, E. and Bridson, R. (2014). Detailed water with coarse grids: combining surface meshes and adaptive discontinuous galerkin. *ACM Trans Graph*, 33(4):136:1–136:9. 37

Enright, D., Marschner, S., and Fedkiw, R. (2002). Animation and rendering of complex water surfaces. *ACM Trans Graph*, 21(3):736–744. 37, 38

Etzmuss, O., Keckeisen, M., and Strasser, W. (2003). A fast finite element solution for cloth modeling. In *Proc. Pac. Graph.*, pages 244–251. 108

Feldman, B., O'Brien, J., and Arikan, O. (2003). Animating suspended particle explosions. *SIGGRAPH '03*, 22(3):708–715. 37

Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graph Mod Imag Proc*, 58:471–483. 37

Gao, Y., Li, C., Hu, S., and Barsky, B. (2009). Simulating gaseous fluids with low and high speeds. *Comp Graph Forum*, 28(28):1845–1852. 37

Gascon, J., Zurdo, J. S., and Otaduy, M. A. (2010). Constraint-based simulation of adhesive contact. In *Proc. Symp. Comp. Anim.*, pages 39–44. 110

Gast, T., Schroeder, C., Stomakhin, A., Jiang, C., and Teran, J. (2015). Optimization integrator for large time steps. *IEEE Trans Vis Comp Graph*, pages 1–1. 5, 96

Gerszewski, D. and Bargteil, A. (2013). Physics-based animation of large-scale splashing liquids. *ACM Trans Graph*, 32(6):185:1–185:6. 37

Gerszewski, D., Bhattacharya, H., and Bargteil, A. (2009). A point-based method for animating elastoplastic solids. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 133–138. 25

Goktekin, T., Bargteil, A., and O'Brien, J. (2004). A method for animating viscoelastic fluids. *ACM Trans Graph*, 23(3):463–468. 24, 58, 59, 72, 89

Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., and Grinspun, E. (2007). Efficient simulation of inextensible cloth. In *ACM Trans. on Graph. (TOG)*, volume 26, page 49. ACM. 95

Gonzalez, M., Schmidt, B., and Ortiz, M. (2010). Force-stepping integrators in lagrangian mechanics. *Intl. J. for Num. Meth. in Engng.*, 84(12):1407–1450. 95

Gonzalez, O. and Stuart, A. (2008). *A First Course in Continuum Mechanics*. Cambridge texts in applied mathematics. Cambridge University Press. 64, 69

Harlow, F. (1964). The particle-in-cell method for numerical solution of problems in fluid dynamics. *Meth Comp Phys*, 3:319–343. 38

Harlow, F. and Welch, E. (1965). Numerical calculation of time dependent viscous flow of fluid with a free surface. *Phys Fluid*, 8(12):2182–2189. 38, 58, 89

Hauth, M. and Etzmuss, O. (2001). A high performance solver for the animation of deformable objects using advanced numerical methods. In *Comp. Graph. Forum*, volume 20, pages 319–328. 94, 95

Hiemenz, P. and Rajagopalan, R. (1997). *Principles of Colloid and Surface Chemistry*. Marcel Dekker. 23

Hirota, G., Fisher, S., Lee, C., Fuchs, H., et al. (2001). An implicit finite element method for elastic solids in contact. In *Comp. Anim., 2001.*, pages 136–254. IEEE. 94, 95, 96

Hirt, C. and Shannon, J. (1968). Free-surface stress conditions for incompressible-flow calculations. *JCP*, 2(4):403–411. 89

Hong, J., Lee, H., Yoon, J., and Kim, C. (2008a). Bubbles alive. *ACM Trans Graph*, 27(3):48:1–48:4. 37

Hong, W., House, D., and Keyser, J. (2008b). Adaptive particles for incompressible fluid simulation. *Vis Comp*, 24(7):535–543. 40

Hong, W., House, D., and Keyser, J. (2009). An adaptive sampling approach to incompressible particle-based fluid. *Theory Pract Comp Graph*, pages 69–76. 40

Ihmsen, M., Cornelis, J., Solenthaler, B., Horvath, C., and Teschner, M. (2013). Implicit incompressible sph. *IEEE Trans Vis Comp Graph*, 20(3):426–435. 37

Irving, G., Teran, J., and Fedkiw, R. (2004). Invertible finite elements for robust simulation of large deformation. In *Proc. 2004 ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 131–140. 72, 108

Iwasaki, K., Uchida, H., Dobashi, Y., and Nishita, T. (2010). Fast particle-based visual simulation of ice melting. *Comp. Graph. Forum*, 29(7):2215–2223. 59, 60

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. (2015). The affine particle-in-cell method. *ACM Trans Graph*, 34(4). 2, 3, 36

Kane, C. (1999). *Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems*. PhD thesis, Caltech. 94, 95

Kane, C., Marsden, J. E., and Ortiz, M. (1999). Symplectic-energy-momentum preserving variational integrators. *J. Math. Phys.*, 40:3353. 95

Kaufman, D. M., Sueda, S., James, D. L., and Pai, D. K. (2008). Staggered projections for frictional contact in multibody systems. In *ACM Trans. Graph. (TOG)*, volume 27, page 164. ACM. 110, 113

Keiser, R., Adams, B., Gasser, D., Bazzi, P., Dutré, P., and Gross, M. (2005). A unified lagrangian approach to solid-fluid animation. In *Eurographics/IEEE VGTC Conf. Point-Based Graph.*, pages 125–133. 25, 59, 60, 65

Kharevych, L., Yang, W., Tong, Y., Kanso, E., Marsden, J., and Schroder, P. (2006). Geometric, variational integrators for computer animation. In *Proc. Symp. Comp. Anim.*, pages 43–51. 95, 109

Kim, J., Cha, D., Chang, B., Koo, B., and Ihm, I. (2006a). Practical animation of turbulent splashing water. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '06, pages 335–344. 38

Kim, T., Adalsteinsson, D., and Lin, M. C. (2006b). Modeling ice dynamics as a thin-film stefan problem. In *Proc 2006 ACM SIGGRAPH/Eurographics Symp Comp Anim*, pages 167–176. 59

Kwatra, N., Su, J., Gretarsson, J., and Fedkiw, R. (2009). A method for avoiding the acoustic time-step restriction in compressible flow. *J. Comp. Phys.*, 228:4146–4161. 70

Larson, R. G. (1999). *The Structure and Rheology of Complex Fluids*. Oxford University Press: New York. 23, 24, 27

Lee, H., Hong, J., and Kim, C. (2009). Interchangeable sph and level set method in multiphase fluids. *Vis Comp*, 25(5):713–718. 37

Lenaerts, T. and Dutre, P. (2009). Mixing fluids and granular materials. *Comp. Graph. Forum*, 28(2):213–218. 59

Lew, A., Marsden, J., Ortiz, M., and West, M. (2004). Variational time integrators. *Intl. J. Num. Meth. Engng.*, 60(1):153–212. 95

Lii, S.-Y. and Wong, S.-K. (2013). Ice melting simulation with water flow handling. *Vis. Comp.*, pages 1–8. 59, 60, 65

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. (2013). Fast simulation of mass-spring systems. *ACM Trans. Graph. (TOG)*, 32(6):214. 94, 95

Losasso, F., Irving, G., Guendelman, E., and Fedkiw, R. (2006a). Melting and burning solids into liquids and gases. *IEEE Trans. Vis. Comp. Graph.*, 12:343–352. 24, 60

Losasso, F., Shinar, T., Selle, A., and Fedkiw, R. (2006b). Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819. 60, 89

Losasso, F., Talton, J., Kwatra, N., and Fedkiw, R. (2008). Two-way coupled sph and particle level set fluid simulation. *IEEE Trans Vis Comp Graph*, 14:797–804. 37

Love, E. and Sulsky, D. (2006). An unconditionally stable, energy-momentum consistent implementation of the the material point method. *Comp Meth App Mech Eng*, 195:3903–3925. 39, 40

Maréchal, N., Guérin, E., Galin, E., Mérillou, S., and Mérillou, N. (2010). Heat transfer simulation for modeling realistic winter sceneries. *Comp. Graph. Forum*, 29(2):449–458. 60, 65

Martin, S., Kaufmann, P., Botsch, M., Grinspun, E., and Gross, M. (2010). Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph.*, 29(4):39:1–39:10. 61

Martin, S., Thomaszewski, B., Grinspun, E., and Gross, M. (2011). Example-based elastic materials. In *ACM Trans. Graph. (TOG)*, volume 30, page 72. ACM. 94, 95, 96

Mast, C., Mackenzie-Helnwein, P., Arduino, P., Miller, G., and Shin, W. (2012). Mitigating kinematic locking in the material point method. *J. Comp. Phys.*, 231(16):5351–5373. 61, 70

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. (2011). Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30:37:1–37:12. 108

Michels, D., Sobottka, G., and Weber, A. (2013). Exponential integrators for stiff elastodynamic problems. In *ACM Trans. Graph. (TOG)*. ACM. 94, 95

Mihalef, V., Metaxas, D., and Sussman, M. (2007). Textured liquids based on the marker level set. *Comp Graph Forum*, pages 457–466. 37

Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574. 68

Morrison, I. and Ross, S. (2002). *Colloidal Dispersions: Suspensions, Emulsions and Foams*. Wiley Interscience. 24

Muller, K., Fedosov, D., and Gompper, G. (2015). Smoothed dissipative particle dynamics with angular momentum conservation. *J Comp Phys*, 281:301–315. 44

Muller, M. and Gross, M. (2004). Interactive virtual materials. In *Proc. Graph. Intl.*, pages 239–246. 108

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. (2005). Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478. 58

Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., and Alexa, M. (2004). Point based animation of elastic, plastic and melting objects. In *ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 141–151. 25, 59

Narain, R., Golas, A., and Lin, M. (2013). Free-flowing granular materials with two-way solid coupling. *ACM Trans Graph*, 29(6):173:1–173:10. 37

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer series in operations research and financial engineering. Springer. 101, 103

Paiva, A., Petronetto, F., Lewiner, T., and Tavares, G. (2006). Particle-based non-newtonian fluid animation for melting objects. In *Conf. Graph. Patt. Images*, pages 78–85. 25, 59, 60, 65

Paiva, A., Petronetto, F., Lewiner, T., and Tavares, G. (2009). Particle-based viscoplastic fluid/solid simulation. *Comp. Aided Des.*, 41(4):306–314. 25, 59, 60, 65

Pandolfi, A., Kane, C., Marsden, J., and Ortiz, M. (2002). Time-discretized variational formulation of non-smooth frictional contact. *Intl. J. Num. Meth. Engng.*, 53:1801–1829. 100

Parks, D. and Forsyth, D. (2002). Improved integration for cloth simulation. In *Proc. of Eurographics*. 95

Patkar, S., Aanjaneya, M., Karpman, D., and Fedkiw, R. (2013). A hybrid lagrangian-eulerian formulation for bubble generation and dynamics. In *Proc ACM SIGGRAPH/Eurograp Symp Comp Anim*, SCA '13, pages 105–114. 38

Prudhomme, R. and Kahn, S. (1996). *Foams: Theory, Measurements, and Applications*. Marcel Dekker. 24

Ram, D., Gast, T., Jiang, C., Schroeder, C., Stomakhin, A., Teran, J., and Kavehpour, P. (2015). A material point method for viscoelastic fluids, foams and sponges. In *Proc ACM SIGGRAPH/Eurographics Symp Comp Anim*, SCA '15. 2, 3

Rasmussen, N., Enright, D., Nguyen, D., Marino, S., Sumner, N., Geiger, W., Hoon, S., and Fedkiw, R. (2004). Directable photorealistic liquids. In *ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 193–202. 24, 60, 89

Raveendran, K., Wojtan, C., and Turk, G. (2011). Hybrid sph. In *Proc 2011 ACM SIGGRAPH/Eurograp Symp Comp Anim*, SCA '11, pages 33–42. 37

Robinson-Mosher, A., Shinar, T., Gretarsson, J., Su, J., and Fedkiw, R. (2008). Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Trans. Graph.*, 27(3):46:1–46:9. 61

Schmedding, R. and Teschner, M. (2008). Inversion handling for stable deformable modeling. *Vis. Comp.*, 24:625–633. 108

Schramm, L. (1994). *Foams: Fundamentals and Applications in the Petroleum Industry*. ACS. 24

Serway, R. A. and Jewett, J. W. (2009). *Physics for Scientists and Engineers*. Cengage Learning. 65

Sifakis, E., Shinar, T., Irving, G., and Fedkiw, R. (2007). Hybrid simulation of deformable solids. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 81–90. 20

Simo, J. C., Tarnow, N., and Wong, K. (1992). Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics. *Computer methods in applied mechanics and engineering*, 100(1):63–116. 95

Sin, F., Bargteil, A., and Hodgins, J. (2009). A point-based method for animating incompressible flow. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 247–255. 37

Solenthaler, B. and Pajarola, R. (2009). Predictive-corrective incompressible sph. In *ACM transactions on graphics (TOG)*, volume 28, page 40. ACM. 61

Solenthaler, B., Schläfli, J., and Pajarola, R. (2007). A unified particle model for fluid-solid interactions. *Comp. Anim. Virt. Worlds*, 18(1):69–82. 25, 58, 59, 60, 65, 89

Song, O., Kim, D., and Ko, H. (2009). Derivative particles for simulating detailed movements of fluids. *IEEE Trans Vis Comp Graph*, pages 247–255. 38

Steffen, M., Kirby, R., and Berzins, M. (2008). Analysis and reduction of quadrature errors in the material point method (MPM). *Int. J. Numer. Meth. Engng*, 76(6):922–948. 72, 74, 88

Stern, A. and Grinspun, E. (2009). Implicit-explicit variational integration of highly oscillatory problems. *Multiscale Modeling & Simulation*, 7(4):1779–1794. 94, 95

Stomakhin, A., Howes, R., Schroeder, C., and Teran, J. (2012). Energetically consistent invertible elasticity. In *ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, pages 25–32. 16, 61, 66, 106, 108, 109

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. (2013). A material point method for snow simulation. *ACM Trans Graph*, 32(4):102:1–102:10. 2, 3, 6, 10, 12, 13, 17, 19, 25, 28, 37, 41, 49, 61, 66, 67, 72, 76, 88, 113, 114, 115, 120

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., and Selle, A. (2014). Augmented mpm for phase-change and varied materials. *ACM Trans Graph*, 33(4):138:1–138:11. 2, 4, 25, 37, 52

Stora, D., Agliati, P.-O., Cani, M.-P., Neyret, F., and Gascuel, J.-D. (1999). Animating lava flows. In *Graph. Int.*, pages 203–210. 59, 60

Su, J., Sheth, R., and Fedkiw, R. (2013). Energy conservation for the simulation of deformable bodies. *Visualization and Computer Graphics, IEEE Transactions on*, 19(2):189–200. 95

Sulsky, D., Zhou, S.-J., and Schreyer, H. (1995). Application of particle-in-cell method to solid mechanics. *Comp. Phys. Comm.*, 87:236–252. 6, 41, 57, 77

Teran, J., Sifakis, E., Irving, G., and Fedkiw, R. (2005). Robust quasistatic finite elements and flesh simulation. In *Proc. Symp. Comp. Anim.*, pages 181–190. 108, 109

Terzopoulos, D. and Fleischer, K. (1988a). Deformable models. *Vis Comp*, 4(6):306–331. 14, 24

Terzopoulos, D. and Fleischer, K. (1988b). Modeling inelastic deformation: Viscolelasticity, plasticity, fracture. *SIGGRAPH Comp Graph*, 22(4):269–278. 14, 24

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically deformable models. *Proc. SIGGRAPH*, 21:205–214. 14

Terzopoulos, D., Platt, J., and Fleischer, K. (1991). Heating and melting deformable models. *J. Vis. Comp. Anim.*, 2(2):68–73. 58, 60

Teschner, M., Heidelberger, B., Muller, M., and Gross, M. (2004). A versatile and robust model for geometrically complex deformable solids. In *Comp. Graph. Int.*, pages 312–319. 59, 60

Um, K., Baek, S., and Han, J. (2014). Advanced hybrid particle-grid method with sub-grid particle correction. *Comp Graph Forum*, 33:209–218. 40

Volino, P. and Magnenat-Thalmann, N. (2001). Comparing efficiency of integration methods for cloth simulation. In *Comp. graph. Intl. 2001 Proc.*, pages 265–272. IEEE. 94, 95

Wang, Y., Jiang, C., Schroeder, C., and Teran, J. (2014). An adaptive virtual node algorithm with robust mesh cutting. In *Proc ACM SIGGRAPH/Eurographics Symp Comp Anim*, SCA '14, pages 77–85. 93

Wei, X., Li, W., and Kaufman, A. (2003). Melting and flowing of viscous volumes. In *Intl. Conf. Comp. Anim. Social Agents*, pages 54–60. 60

Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., and O'Brien, J. F. (2010). Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics*, 29(4):49:1–11. Proc. of ACM SIGGRAPH 2010. 59

Wojtan, C., Carlson, M., Mucha, P. J., and Turk, G. (2007). Animating corrosion and erosion. In *Eurographics Conf. Nat. Phen.*, pages 15–22. 60

Wojtan, C., Thürey, N., Gross, M., and Turk, G. (2009). Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3):76:1–76:10. 24, 59

Wojtan, C. and Turk, G. (2008). Fast viscoelastic behavior with thin features. *ACM Trans. Graph.*, 27(3):47:1–47:8. 24, 59

Yabe, T., Xiao, F., and Utsumi, T. (2001). The constrained interpolation profile method for multiphase analysis. *J Comp Phys*, 169:556–593. 38

Yu, J. and Turk, G. (2010). Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proc. of the 2010 ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, pages 217–225. Eurographics Association. 90

Yue, Y., Smith, B., Batty, C., Zheng, C., and Grinspun, E. (2015). Continuum foam: A material point method for shear-dependent flows. *To appear, ACM Trans Graph*. 2, 25, 35

Zhao, Y., Wang, L., Qiu, F., Kaufman, A., and Mueller, K. (2006). Melting and flowing in multiphase environment. *Comp. Graph.*, 30:2006. 60

Zheng, C. and James, D. L. (2011). Toward high-quality modal contact sound. In *ACM Transactions on Graphics (TOG)*, volume 30, page 38. ACM. 113

Zhu, B., Yang, X., and Fan, Y. (2010a). Creating and preserving vortical details in sph fluid. *Comp Graph Forum*, 29(7):2207–2214. 37

Zhu, Y. and Bridson, R. (2005). Animating sand as a fluid. *ACM Trans Graph*, 24(3):965–972. 37, 40, 41, 57

Zhu, Y., Sifakis, E., Teran, J., and Brandt, A. (2010b). An efficient and parallelizable multigrid framework for the simulation of elastic solids. *ACM Trans. Graph.*, 29:16:1–16:18. 108