Title
Animating Viscoelastic Fluids

Author
Goktekin, Tolga Gokce

Publication Date
2011

Peer reviewed|Thesis/dissertation

Animating Viscoelastic Fluids

by

Tolga Gokce Goktekin

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

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James F. O'Brien, Chair
Professor Jonathan Shewchuk
Professor Panayiotis Papadopoulos

Fall 2011

Animating Viscoelastic Fluids

Abstract

Animating Viscoelastic Fluids

by

Tolga Gokce Goktekin

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor James F. O'Brien, Chair

In this thesis we describe a technique for animating the behavior of viscoelastic fluids such as mucus, liquid soap, pudding, toothpaste, or clay, that exhibit a combination of both fluid and solid characteristics. The technique builds upon prior Eulerian methods for animating incompressible fluids. Our method computes viscoelastic fluid behavior by supplementing the basic Navier–Stokes equations with additional terms for elastic body forces. These terms can be readily computed on rectilinear grids using a staggered discretization scheme, and the use of an Eulerian formulation easily accommodates modeling flows that undergo large deformations with topological changes. These elastic terms require computing the material strain throughout the fluid. Because the fluid simulations do not make use of an explicit reference configuration, strain is computed by integrating strain rate and advecting the results. The transition from elastic resistance to viscous flow is controlled by von Mises' yield condition, and subsequent behavior is governed by a quasi-linear plasticity model.

To my wife Yeşim, and my sons Ege & Berge

Contents

List of Figures

Acknowledgments

It took me a lot of years from the point of my first visiting scholar position at Berkeley to the point when I finally finished my dissertation. I am grateful to a lot of people who supported me all these years and helped me realizing my dreams. I would like to acknowledge them all here.

First and foremost, I would like to thank my advisor extraordinaire and friend James O'Brien. It was under his mentorship that I finally worked on something that I truly believed in after a long past in higher education. He has the magic of making the complex trivial and intuitive, which helped me tremendously at the times when I was buried deep in the theory. Thanks to him, I am finally *3D*.

I also would like to thank the rest of my committee members, Jonathan Shewchuk and Panayiotis Papadopoulos, for their ideas, support and encouragement for all these years.

In my research I had the privilege working with a great many talented people. I liked to thank all the members of the B-CAM group for their friendship and support. In particular I'd like to thank my co-author and dear friend Adam Bargteil, for his contributions to my research, especially implementing the particle level-set method. I am also grateful to my dear friend Okan Arikan and his magical renderer Pixie.

I would like to thank Shankar Sastry who supported me in my early years in Berkeley. He gave me the opportunity to work in many interesting projects and helped me pursuing my passion for computer graphics.

For the last six years I have been working as a visual effects artist in the wondrous Pixar Animation Studios, where I applied the ideas and methods I present here to various effects that I created. I am especially grateful to Apurva Shah for mentoring me in my early years at Pixar and for helping me finding the artist in myself.

Finishing the writing of this thesis took way longer than expected. To this end, I owe a special gratitude to my wife Yeşim for her patience, perseverance and tremendous support for helping me finish my dissertation. Through this, she has become an expert on viscoelastic flow and she is forever scarred to look at every viscoelastic fluid with a different eye no matter how mundane it is. She now joins me exclaiming how low or high its elastic decay rate is every time we switch to a different shampoo.

Chapter 1

Introduction

A fluid that's macromolecular
Is really quite weird — in particular
The abnormal stresses
The fluid possesses
Give rise to effects quite spectacular.

R. B. Bird

In our everyday lives we are surrounded with many different materials that behave in varied and complex ways. One of the major challenges facing the computer graphics researchers and the visual effects artist is animating these materials as realistically as possible. Over the years numerous methods have been developed and successfully utilized for animating fluids and solids of various scales. However these methods tend to simplify the wide ranging spectrum of behavior exhibited by these real life materials into two distinct idealizations: simple Newtonian fluids and elastic solids. While this classification of solid or fluid is valid for some materials such as water, smoke, rubber or cloth, it is insufficient for a great many others. In fact, there is a large family of natural and artificial materials that exhibit a combination of both fluid and solid behavior. Often referred to as *viscoelastic fluids*, these materials initially respond to strain elastically like a solid, but subjected to increasingly large stresses, they flow like a fluid. A few common examples include egg white, dough, mayonnaise, mucus, clay, unset cement, liquid acrylic, gelatin, gels, liquid soap, and toothpaste. Like an elastic solid, viscoelastic fluids can bounce and jiggle, but they can also flow like a fluid. For some of these viscoelastic fluids, such as mucus or egg white, the combination of elastic and fluid behavior is quite apparent. For others, such as liquid soap, the fluid behavior dominates due to the lack of strong elastic forces within the fluid and differs only subtly from that of a purely viscous fluid. Finally, other viscoelastic fluids, such as gels, dough or mayonnaise, behave mostly elastically under simple gravity and flow only if they are subjected to substantial stress.

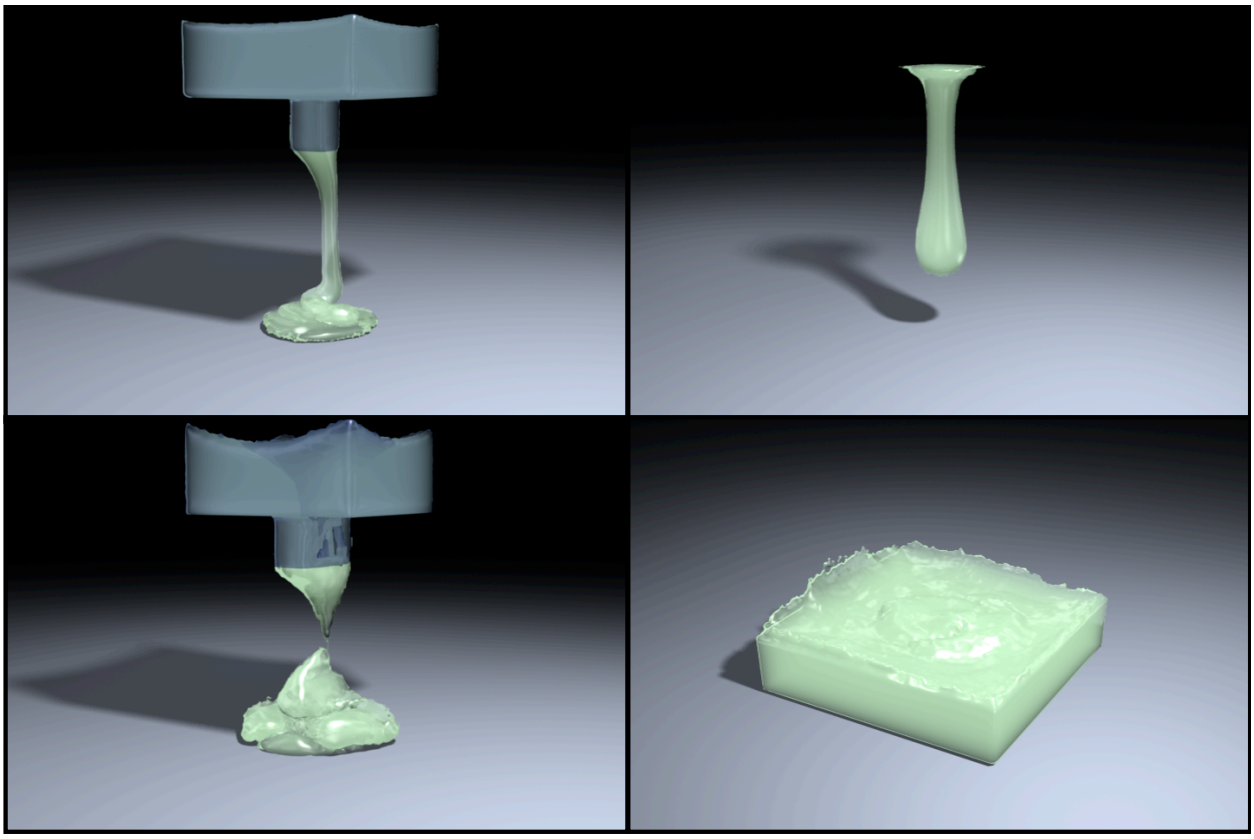

Figure 1.1: A few examples produced by our method demonstrating different behaviors of viscoelastic fluids.

This combined fluid and elastic nature of the viscoelastic fluids gives rise to an extensive set of interesting behaviors. A viscoelastic fluid, such as liquid shampoo, can fold on itself or swirl as it pours down from a bottle. Mayonnaise easily spreads like a viscous fluid forced by a knife, but holds its shape indefinitely when the force is removed, thus presenting solid behavior. A hair gel flows out as bouncy chunks from its container. Yet, none of these viscoelastic behaviors can be captured by a simulation method that is solely designed for a Newtonian fluid or an elastic solid. In this thesis, we provide a method for animating viscoelastic fluids which captures fundamental characteristics of viscoelastic flow (see Figure 1.1).

Viscoelastic materials owe their distinct behavior to their composition. They are primarily made up of *macromolecules* (*e.g.* polymers) suspended or solved in a Newtonian fluid. The various types of bonds within and/or between these large molecules create *microscale* spring-like forces in the material. At *macroscale*, the aggregate of these molecular level forces causes the material behave elastically and resist shape change, while flowing like a fluid. If the amount of deformation in the material, measured by strain, is below a limit, called *yield point*, these elastic forces dominate and cause the viscoelastic fluid behave just like an elastic

solid. However, if the strains exceed the yield point, the material starts to flow plastically and the elastic forces start decaying over time, with a rate called *elastic decay rate*. This gradual elastic decay, also known as *creep*, is a fundamental property of viscoelastic fluids. The two parameters, yield point and elastic decay rate, span the space between solid and fluid materials. If the yield point is set to a high value or the decay rate is set to zero, the material behaves like an elastic solid. If, however, the yield point is set to zero and the decay rate is set to a high value, the material behaves like a fluid. Intermediate values correspond to viscoelastic fluids.

1.1 Modeling Viscoelastic Fluids with Yielding

The study of viscoelastic behavior in mechanical engineering dates back to Maxwell's seminal work in 1867, in which he formulated the first model of a *general linear viscoelastic fluid* [Maxwell, 1867]. In this formulation the elastic forces in the fluid are modeled as springs obeying Hook's law where the elastic forces depend linearly on the strain that builds up in the fluid. As the fluid flows, these elastic forces decay exponentially causing creep in the material. This simple model, often referred to as the *Maxwell fluid*, serves as the basic building block in the study of viscoelastic fluids [Bird *et al.*, 1987a]. Yet, Maxwell's original model and most of the subsequent extensions to it do not have a yield point that defines the transition from elastic solid into viscoelastic flow.

The concept of plastic yielding was first introduced in early 1900s in solid mechanics to model the behavior of metals under high stresses [Osakada, 2010]. In 1913, von Mises formulated a yield criterion where a highly stiff elastic solid like steel transitions into a highly viscous fluid, if the deviatoric part of the stress tensor which defines the distortion in the material exceeds a material dependent limit [von Mises, 1913]. In fluid dynamics, the first application of a yield criteria was formulated by Bingham to model the behavior of suspension of finely divided solid materials in fluids such as pastes, paint and clay in water [Bingham, 1922]. In this one-dimensional model, often referred as *Bingham fluid* or *viscoplastic fluid*, a rigid solid transitions into a viscous fluid, if the sheer stresses exceed a limit, called *yield stress*. In 1932, using von Mises' yield criterion, Hohenemser et al. [1932] generalized the simple Bingham model into three-dimensions. Subsequently, Oldroyd [1947] proposed a three dimensional model which combines the von Mises' yield criterion together with a linear elastic solid before yielding and a viscous fluid after yielding. Over the years, models that extended upon Bingham fluids consistently assumed that the behavior after yield point is of a purely viscous fluid. In all these formulations the plastic flow occurs instantaneously resulting in purely viscous flow which fails to model the notion of creep fundamental to viscoelastic flow.

The model we present in this thesis combines von Mises' yield criteria found in more recent formulations of Bingham fluids with the Maxwell's model of viscoelastic flow to animate materials which transition from elastic solids into viscoelastic flow. In particular, we compute

elastic forces in the fluid based on the elastic strains that develop in the material. If the deviatoric part of the elastic strain exceeds the yield point, we start decaying the elastic strain with plastic flow rate using Maxwell's model. It's important to note that our model generalizes both Maxwell fluids and Bingham fluids. If we set the yield point to zero, our model reduces to a Maxwell fluid. If we set the plastic flow rate to a high value, we obtain Oldroyd's formulation of Bingham fluids where an elastic solid transitions into purely viscous fluid. All other combinations model wide ranging spectrum of materials that behave elastically before yielding and flow viscoelastically afterwards.

1.2 Overview of the Proposed Method

The technique we present for simulating viscoelastic fluids builds on prior Eulerian grid–based methods for animating incompressible fluids with free surfaces. As evidenced by their widespread use, these methods can efficiently produce results that are realistic enough for applications in the visual effects industry. The viscoelastic fluid behavior is simulated by supplementing the basic Navier–Stokes equations with additional terms for elastic body forces. These elastic forces can be readily computed on rectilinear grids using a staggered discretization scheme, and the use of an Eulerian formulation easily accommodates modeling flows that undergo large deformations and topological changes. These elastic terms require computing the material strain tensor throughout the fluid. Because the fluid simulations do not make use of an explicit reference configuration, strain is computed by integrating strain rate and advecting the results. The transition from elastic resistance to viscoelastic flow is controlled by a user-specified yield point parameter based on von Mises' yield condition. Once the strain in the fluid exceeds this point, the elastic strain starts to decay by a user-specified rate, resulting in a quasi-linear plasticity model.

The Eulerian framework we use for simulating viscoelastic fluids consists of three parts that work together to produce useful results. First, we store the pressure, velocity and strain values that define the fluid's state on a rectilinear regular grid. To avoid spurious oscillations in the pressure and velocity fields which can lead to instability, we use the straggered grid scheme introduced by Harlow and Welch [1965] for storing scalar pressure and vector velocity values. Additionally, we extend their scheme to store the strain tensor field where we put the diagonal elements of strain at cell centers and off-diagonal elements at edge centers. This 3D staggered tensor scheme is also an extension of the 2D scheme proposed by Gerittsma [1996].

Second, we discretize and solve the two principal partial differential equations of our system, the modified Navier–Stokes and the strain update, term-by-term on the grid using the finite differences method. We first compute the elastic forces due elastic strain and integrate these forces using explicit forward Euler time stepping scheme. Then, we solve the standard Navier–Stokes equations using a series of methods introduced by Stam et al. [1999]; we advect the velocity using the semi-Lagrangian advection method, integrate viscous forces using the implicit backward Euler scheme and impose incompressibility using the pressure

projection method. Finally, we update the strain tensor where we advect the strain using the Semi-Lagrangian advection method, integrate the strain rate, and, if it exceeds the yield point, we decay the strain using explicit forward Euler scheme.

The third component of our system is the tracking of the free surface of the fluid where we use the particle levelset surface tracking method described by Enright et al. [2004]. We first express the fluid surface as an implicit function whose levelset at zero locates the boundary of the fluid. This levelset function is represented using a combination of particles and signed distance values stored on a second regular rectilinear grid. Once we compute our final velocity, we advect these particles and the signed distance values to advance the levelset function that defines the fluid surface.

1.3 Related Work

In the field of computer graphics, the technique described by Carlson et al. [2002] is possibly closest in intent to the method we describe here. Like us, they were interested in modeling materials with properties intermediate between solids and fluids using an Eulerian grid-based fluid simulation method. However, they opt to map the continuum between fluids and solids to varying viscosity. In their system a solid is simply a fluid with very high viscosity. This approach ignores the elastic behavior demonstrated by many materials. Nevertheless, they do generate nice results for highly viscous fluids, and they describe an implicit integration method for coping with stability issues arising from very high viscosities.

Other graphics researchers have used particle-based methods for modeling highly viscous fluids and fluids with some form of elasticity. In [Terzopoulos *et al.*, 1989] the authors modeled melting thermoelastic materials. The particles exerted cohesive, viscous, and volume-preserving forces on their neighbors. While solid, each particle was connected to a fixed set of neighbors using elastic springs. As the material would become more fluid-like, the springs would weaken, and eventually disappear. By varying the elastic properties of the materials, this method could model a range of behaviors, but without plasticity, it could not model materials, like clay, that flow into a new configuration and then resist changes from that configuration. Similar approaches using different particle formulations have appeared in [Desbrun and Gascuel, 1995],[Desbrun and Cani, 1996],[Cani and Desbrun, 1997], and [Stora *et al.*, 1999]. The method appearing in [Desbrun and Gascuel, 1995] used elastic forces with dynamically determined neighbors to allow behavior that is similar to plastic flow.

Perhaps the greatest limitation on the level of realism achievable by these particle methods was the relatively small number of particles used. However, as processor speeds have increased, particle-based methods have been able to achieve increasingly impressive results. Compelling real-time results for modestly sized systems appear in [Müller *et al.*, 2003], and [Premoze *et al.*, 2003] demonstrates off-line results that are comparable to the current best grid-based methods. Although both of these recent methods focus on strictly liquid behavior, they could be extended along lines similar to what we propose here.

Some methods for modeling solids have dealt with limited amounts of plastic flow. Both [Terzopoulos and Fleischer, 1988a] and [Terzopoulos and Fleischer, 1988b] describe transition to plastic flow based on von Mises's yield condition, and [O'Brien *et al.*, 2002] used a similar plasticity model for ductile fracture behavior. We use the same yield condition of von Mises, but we do not assume that plastic flow occurs instantaneously. Instead we use a more complex model that accommodates phenomena such as creep. Additionally, these prior methods used Lagrangian meshes with largely fixed topology, and so they would have encountered "tangling" difficulties, such as inverting elements, for large amounts of plastic flow.

Another, rather interesting, approach to combining solid and fluid behaviors appears in [Nixon and Lobb, 2002]. They surround a fluid simulation with an elastic membrane. The result is an object that behaves somewhat like a water balloon.

Our work builds directly on previous grid-based, Eulerian methods for animating fluids with free surfaces. Details on these methods can be found in [Foster and Metaxas, 1996], [Stam, 1999], [Foster and Fedkiw, 2001], and [Enright *et al.*, 2002]. In particular, our work essentially extends [Enright *et al.*, 2002] to include the behavior of viscoelastic fluids.

Outside the graphics field, viscoelastic materials have been studied extensively. We refer the reader to the texts [Fung, 1965], [Han and Reddy, 1999], and [Bird *et al.*, 1987a] for detailed descriptions of several different models for viscoelastic and elastoplastic materials. Some recent examples of fluid simulation outside the graphics literature that involve elastic forces include [Gerritsma, 1996], [Tomé *et al.*, 2002], and [Bonito *et al.*, 2006]. However, none of the viscoelasticity models used in these examples include plastic yielding. A detailed analysis of two-dimensional simulations of viscoelastic fluids on staggered rectilinear grids appears in [Gerritsma, 1996]. The three-dimensional method we use for storing rank-two tensor quantities on a staggered grid is a generalization of their two-dimensional method. In [Bonito *et al.*, 2006] a combination of rectilinear grids and finite elements are used with a volume-of-fluid method to model three-dimensional fluids with elastic properties. They store all quantities including vector and tensor components at cell centers. The marker-and-cell (MAC) based method in [Tomé *et al.*, 2002] is another example for solving viscoelastic free-surface flows, where they address issues relating to elastic-stress based boundary conditions at rigid-body and free-surfaces. They use a staggered grid for the velocity field, but the tensor values are collocated at the cell centers.

1.4 Contributions

In this thesis we propose a novel method for animating viscoelastic fluids. When we design our material models and methods to simulate them, we are making several assumptions. First, our main goal is to provide an efficient and easy-to-use tool for the visual effects artists to generate believable animations of a wide range of materials. Therefore, we make no particular attempt to model a specific type of a viscoelastic fluid, nor we strive to achieve

high levels of numerical accuracy that would be required by most engineering applications. Second, our main focus is to animate the complex behavior on the fluid end of the solid-fluid spectrum. Therefore, we intentionally base our method on an Eulerian based formulation. In simulating purely elastic solids, the strain rate based formulation presented in this thesis would cause considerable drift.

The specific contributions of this thesis to the field of computer graphics are as follows:

- It introduces a novel model for animating a material that transitions from an elastic solid into a viscoelastic flow
- The behavior is controlled by a concise set of parameters: plastic yield point which defines when the material transitions from elastic solid into a viscoelastic fluid, and elastic decay rate which defines how fast the elastic forces decay over time. With the use of these two intuitive parameters the artist can animate materials ranging from elastic solids to viscous fluids and viscoelastic fluids in-between.
- It presents a simple and feasible add-on tool to an existing Eulerian based fluid simulator Additional viscoelastic computations introduce only about %7 overhead.
- The method is flexible enough to work with various different fluid simulation techniques (e.g. particle based methods such as Smoothed-Particle Hydrodynamics (SPH) or hybrid particle-grid based methods such as Fluid-Implicit-Particle (FLIP)). All the underlying fluid simulator needs to provide is the support for user-specified internal forces and the ability to report the velocity of the fluid and the distance to the free surface at a given point in space.
- The Eulerian framework provides a robust method for simulating large, complex deformations of viscoelastic fluids with topological changes.

Since its first publication in [Goktekin *et al.*, 2004], the method presented in this thesis has proven to be very production friendly in the visual effects industry. It was successfully used to animate various food effects in the Pixar Animation Studios' *Ratatouille* (see Figure 1.2) [Goktekin *et al.*, 2007]. To our knowledge, as of this writing, our method has been implemented in various commercial or open source Eulerian grid-based fluid simulators including SideFx's Houdini (www.sidefx.com) and Physbam (physbam.stanford.edu). In addition, it also had been implemented in Dreamwork/PDI's in-house fluid simulator to animate the honey effects in *Bee Movie* [Ruilova, 2007].

The following chapters detail these contributions and the results obtained from their implementation. The details of supplementing the governing equations of a fluid, the strain integration and the plasticity model are discussed in Chapter 2. The discretization of the resulting governing equations and details of the proposed viscoelastic fluid simulator are detailed in Chapter 3. Finally, the results by the techniques presented here are discussed in Chapter 4.

Figure 1.2: A couple of viscoelastic food effects from Ratatouille: (left) A dough being prepared in a bowl, (right) A viscoelastic sauce being poured on a dish.

Chapter 2

Modeling Viscoelastic Fluids

The main characteristics of a viscolelastic fluid is the fact that at macroscale, it behaves like a combination of an elastic solid and a simple fluid. In microscale, the mechanism that causes the elastic behavior in a given viscoelastic fluid can vary from one material to another. In some viscoelastic materials, called *polymeric liquids*, the elastic behavior is due to the forces within or between the large, potentially tangled polymer molecules which act as springs inside the fluid. In some other materials, called *polyelectrolytes*, electrostatic forces between these large polymer molecules can cause resistance to shape change, resulting in elastic behavior. These molecular level, microscale differences between viscoelastic fluids may cause subtle differences in behavior that need to be captured by specific engineering applications. Bird et al. [1987b] provide an extensive set of detailed discussions on various viscoelasticity models motivated by some of these microscale mechanisms.

In this thesis our focus is to capture the characteristic behavior of viscoelastic fluids at the macroscale. At this scale, regardless of the exact mechanism, all viscoelastic fluids contain elastic forces within the fluid. This causes the fluid resist shape change as it flows and results in the combined elastic and fluid behavior. In order to understand the behavior of viscoelastic fluids and further define the mechanisms that transform a viscoelastic fluid from one end of the spectrum to the other, the basic definitions of simple fluids and elastic solids need to be reviewed.

2.1 Solids, Fluids and the In-between

The fundamental distinction between an ideal elastic solid and a simple Newtonian fluid is how they respond to deformation caused by an external loading. An ideal elastic solid is a material that cannot be deformed permanently. It always remembers its original shape. If it is deformed by a load, it will go back to its original configuration upon unloading (Figure 2.1). All of the initial deformation will be reversed. An elastic solid can possess viscosity, which affects the speed of the recovery and the amount it oscillates around the rest configuration.

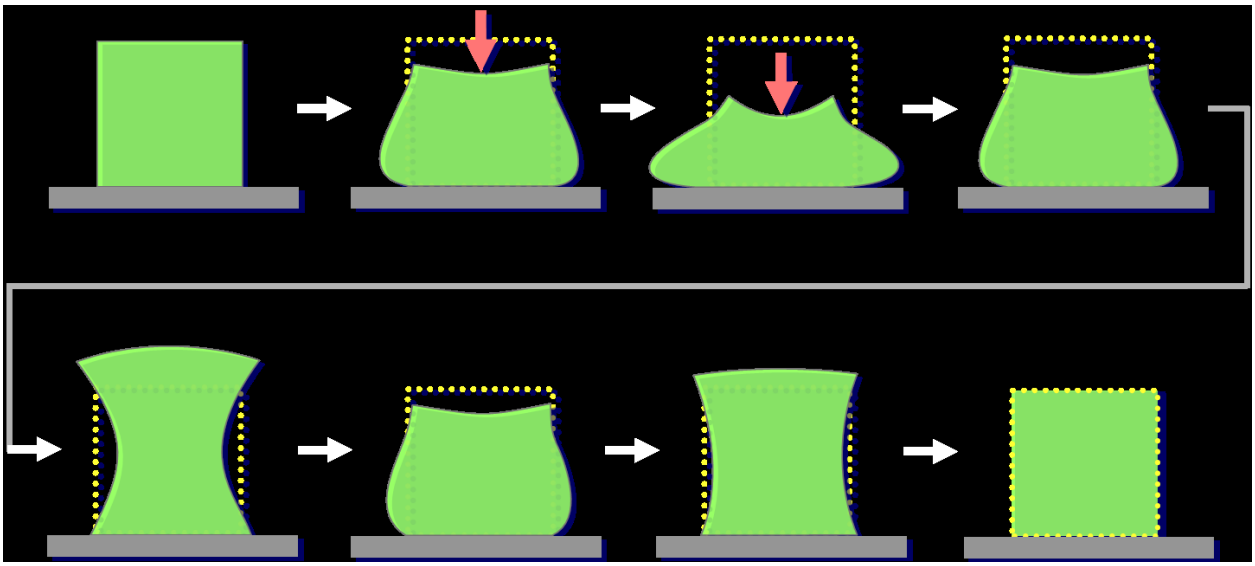

Figure 2.1: Behavior of an elastic solid upon unloading: It oscillates and eventually recovers its original shape.

No matter how viscous an elastic solid is, it always recovers its original shape.

A simple Newtonian fluid represents the other end of the spectrum. It represents materials where every deformation is permanent. Fluids flow with the slightest loading until they fully conform to the shape of their container, and they never recover their original shape (Figure 2.2). The viscosity of a fluid only changes the rate of its flow. A fluid with high viscosity flows slower, but it does not behave elastically.

A viscoelastic fluid has two regimes. If the amount of deformation is below a threshold, called the *plastic yield point*, a viscoelastic fluid will behave just like an elastic solid and recover its original shape upon unloading. If the deformation exceeds the plastic yield point, some of the deformation will be recovered, but the rest will be permanent, i.e. the fluid will deform plastically (Figure 2.3). This plastic deformation will not be instantaneous. Instead it will grow over time at some rate, called the *elastic decay rate*. This behavior is called *plastic flow* or *creep* (Figure 2.4). Along with the viscosity of the fluid, the plastic yield

Figure 2.2: Behavior of a simple fluid under load: It flows amorphously and creates a puddle on the ground.

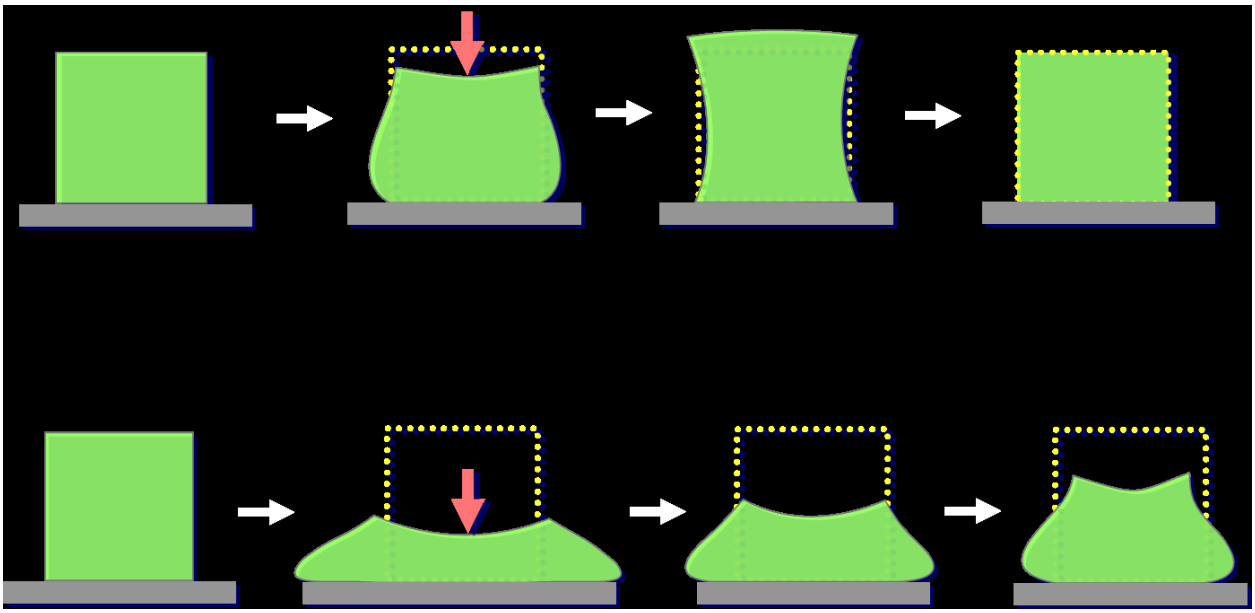

Figure 2.3: Two regimes of a viscoelastic material: (top) elastic response, (bottom) plastic flow.

point and elastic decay rate are the main material parameters that define the behavior of a viscoelastic fluid.

2.2 Governing Equations

In continuum mechanics, the dynamics of materials are governed by three balance laws that every material must obey: *balance of linear momentum*, *balance of mass* and *balance of angular momentum*. The balance of linear momentum states that at any point in a material, the material's acceleration is

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\left(\boldsymbol{u}\cdot\boldsymbol{\nabla}\right)\boldsymbol{u} + \frac{1}{\rho}\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \frac{\boldsymbol{f}}{\rho} \quad , \tag{2.1}$$

where $\boldsymbol{u}$ is the velocity vector of the material at a point in space, ρ its density, $\boldsymbol{\sigma}$ the 3x3 stress tensor at that point of space, and $\boldsymbol{f}$ represents all other force vectors such as gravity. The symbol $\boldsymbol{\nabla}$ denotes the vector of differential operators $\boldsymbol{\nabla} = [\partial/\partial x, \partial/\partial y, \partial/\partial z]^{\mathsf{T}}$.

The balance of mass states that at any point in a material, the rate in which the mass enters is equal to the rate at which the mass leaves, which can be expressed as

$$\frac{\partial \rho}{\partial t} + \rho\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 \quad . \tag{2.2}$$

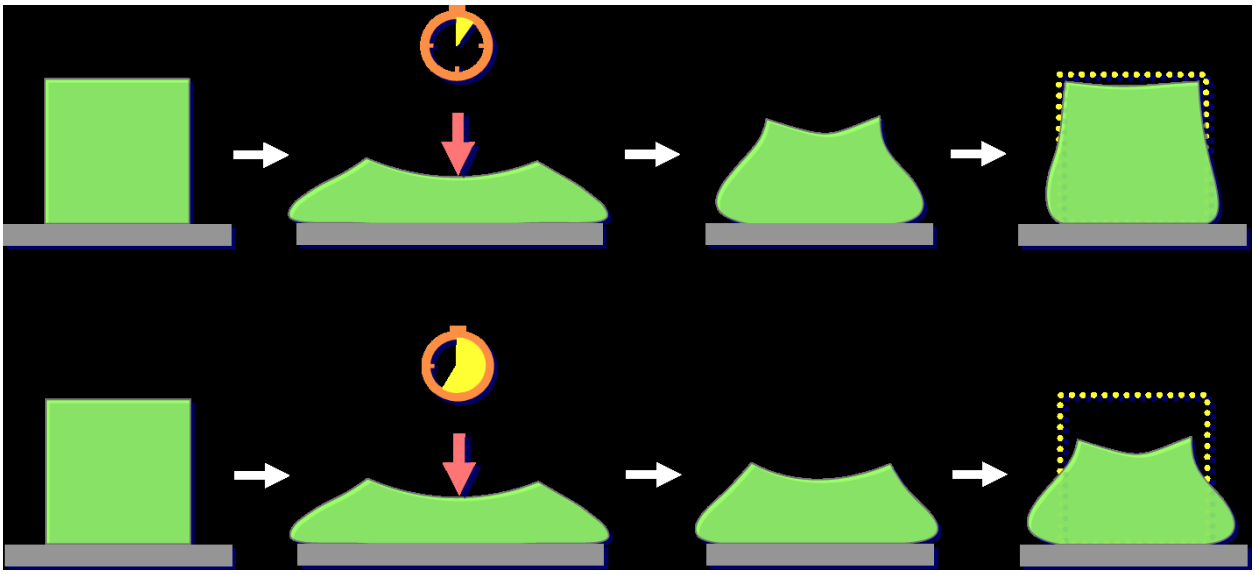

Figure 2.4: Two viscoelastic bodies undergoing same deformation with different durations: (top) Force is applied for a short time and the object recovers most of its original shape, (bottom) Same force is applied for longer amount of time and the object recovers less of its original shape.

For incompressible materials, where $\partial\rho/\partial t = 0$, Equation (2.2) reduces to

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \quad . \tag{2.3}$$

Finally the balance of angular momentum requires that the stress tensor is symmetric

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{\mathsf{T}} \quad . \tag{2.4}$$

To drive the governing equations for a specific material, we need to define the stress as a function of how that specific material responds to deformation. In other words, we need to formulate the *constitutive equation* of that material. Since viscoelastic fluids act as a combination of a fluid and an elastic solid, we can decompose the stress in a viscoelastic fluid into a fluid and an elastic part

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_F + \boldsymbol{\sigma}_E \quad . \tag{2.5}$$

The stresses in an incompressible Newtonian fluid are composed of isotropic pressure which resists volume change in the material, and viscosity of the fluid which acts as an internal friction to shear motion

$$\boldsymbol{\sigma}_F = -p\boldsymbol{I} + \mu_v \left(\boldsymbol{\nabla}\, \boldsymbol{u} + (\boldsymbol{\nabla}\, \boldsymbol{u})^{\mathsf{T}} \right) \quad , \tag{2.6}$$

where $\boldsymbol{I}$ is the 3x3 identity tensor and μ_v is the coefficient of viscosity of the fluid.

The stresses in a general Hookean elastic solid depend linearly on the strain and can be formulated as

$$\boldsymbol{\sigma}_E = \boldsymbol{C}\boldsymbol{\epsilon} \quad , \tag{2.7}$$

where $\boldsymbol{C}$ is a material dependent, fourth order (i.e 3x3x3x3) coefficient tensor, and $\boldsymbol{\epsilon}$ is the 3x3 symmetric strain tensor. If we assume that the elastic material is isotropic, which requires $\boldsymbol{C}$ to be symmetric in all directions, and incompressible, i.e. the material's dilation $tr(\boldsymbol{\epsilon})$ is zero, the general form in Equation (2.7) reduces to

$$\boldsymbol{\sigma}_E = \mu_e\boldsymbol{\epsilon} \quad , \tag{2.8}$$

where μ_e is the elastic modulus of the material.

If we combine the constitutive equations of a Newtonian fluid in Equation (2.6) and an elastic solid in Equation (2.8) using Equation (2.5), we obtain our constitutive equation for an incompressible viscoelastic fluid

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \mu_v\left(\boldsymbol{\nabla}\,\boldsymbol{u} + (\boldsymbol{\nabla}\,\boldsymbol{u})^{\mathsf{T}}\right) + \mu_e\boldsymbol{\epsilon} \quad . \tag{2.9}$$

Putting this constitutive equation into the balance of linear momentum in Equation (2.1), we obtain the equation that governs the behavior of the viscoelastic fluids. Assuming μ_v and μ_e are homogeneous throughout the material, at any point in the fluid, the fluid's acceleration is

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\left(\boldsymbol{u}\cdot\boldsymbol{\nabla}\right)\boldsymbol{u} - \frac{\boldsymbol{\nabla}p}{\rho} + \frac{\mu_v}{\rho}\boldsymbol{\nabla}^2\boldsymbol{u} + \frac{\mu_e}{\rho}\boldsymbol{\nabla}\cdot\boldsymbol{\epsilon} + \frac{\boldsymbol{f}}{\rho} \quad , \tag{2.10}$$

subject to the constraint that the fluid is incompressible

$$\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 \quad . \tag{2.11}$$

The formula in Equation (2.10) is a modified version of the Navier–Stokes equations that includes the additional term $\mu_e/\rho\boldsymbol{\nabla}\cdot\boldsymbol{\epsilon}$, which computes the macroscale elastic forces acting on the fluid.

2.3 Elastic, Plastic, and Total Strain

The fourth term of Equation (2.10) computes the acceleration due to the elastic forces. It requires knowing the elastic strain throughout the fluid. If we had an explicit deformation function, then we could use its spatial derivatives to compute the strain. However, with the Eulerian formulation we are using, there is no deformation function available. Furthermore, the large deformation and flow experienced by the material makes tracking deformation impractical. Instead, we compute the strain by integrating the strain rate. In addition, because

viscoelastic fluids are not perfectly elastic materials, we must observe rules concerning how the elastic strain changes due to plastic yielding.

We first separate the total strain into an elastic and a plastic component so that

$$\boldsymbol{\epsilon}^{\text{Tot}} = \boldsymbol{\epsilon}^{\text{Elc}} + \boldsymbol{\epsilon}^{\text{Plc}} \quad , \tag{2.12}$$

where $\boldsymbol{\epsilon}^{\text{Tot}}$, $\boldsymbol{\epsilon}^{\text{Elc}}$, and $\boldsymbol{\epsilon}^{\text{Plc}}$, are respectively the symmetric total, elastic, and plastic strain tensors. (Outside this section elastic strain is simply denoted as $\boldsymbol{\epsilon}$.) The strain rate is the time derivative of the total strain, therefore the total strain at some time t is

$$\boldsymbol{\epsilon}^{\text{Tot}} = \boldsymbol{\epsilon}_0^{\text{Tot}} + \int_0^t \dot{\boldsymbol{\epsilon}}^{\text{Tot}} \, \mathrm{d}t \quad , \tag{2.13}$$

where $\boldsymbol{\epsilon}_0^{\text{Tot}}$ is the total strain at time $t = 0$ and $\dot{\boldsymbol{\epsilon}}^{\text{Tot}}$ is the strain rate given by[1]

$$\dot{\boldsymbol{\epsilon}}^{\text{Tot}} = \frac{\boldsymbol{\nabla}\, \boldsymbol{u} + (\boldsymbol{\nabla}\, \boldsymbol{u})^{\mathsf{T}}}{2} \quad . \tag{2.14}$$

Similarly, the plastic strain is given by integrating plastic flow.

$$\boldsymbol{\epsilon}^{\text{Plc}} = \boldsymbol{\epsilon}_0^{\text{Plc}} + \int_0^t \dot{\boldsymbol{\epsilon}}^{\text{Plc}} \, \mathrm{d}t \tag{2.15}$$

We use von Mises' criterion for determining when plastic flow should occur. This criterion depends on the elastic strain deviation $\boldsymbol{\epsilon}'$, which is the elastic strain with any dilation removed[2]

$$\boldsymbol{\epsilon}' = \boldsymbol{\epsilon}^{\text{Elc}} - \frac{\text{Tr}\left(\boldsymbol{\epsilon}^{\text{Elc}}\right)}{3} \boldsymbol{I} \quad . \tag{2.16}$$

So long as the magnitude (Frobenius norm) of the strain deviation remains below the yield point γ, no plastic flow occurs. When the limit is exceeded, flow occurs at a rate proportional to the amount by which he limit has been exceeded. So the flow rate for plastic strain is

$$\dot{\boldsymbol{\epsilon}}^{\text{Plc}} = \alpha \frac{\boldsymbol{\epsilon}'}{\|\boldsymbol{\epsilon}'\|} \max\left\{0, \|\boldsymbol{\epsilon}'\| - \gamma\right\} \quad , \tag{2.17}$$

[1]Note that the strain rate in Equation (2.14), also called *rate of stretch tensor*, only measures the rate of the deformation in the material and vanishes for rigid body motion. Even though the closest strain metric whose derivative yields the rate in Equation (2.14) is the geometrically linear *Cauchy's strain*, the strain computed by integrating Equation (2.14) does not create spurious strains for rigid body rotation and thus, unlike Cauchy's strain, is invariant under rigid body rotations. However, as we will see in Section 2.4, we can still break the rotational invariance property, if we don't pay special attention to how we integrate the strain rate while advecting it through the material using our Eulerian formulation.

[2]This distinction is a bit pedantic here, because the fluid is incompressible and thus the dilation should always be zero.

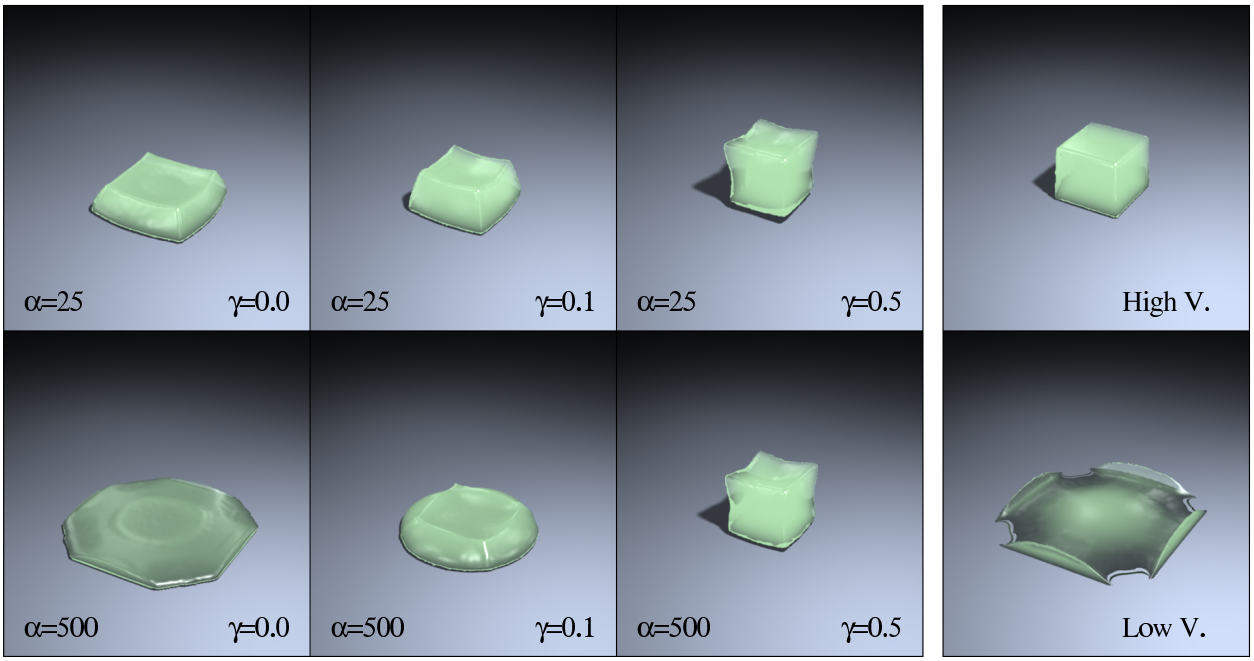

Figure 2.5: This figure demonstrates some effects achieved by varying elastic decay rate α and plastic yield point γ. Each image shows a cube of material that has been dropped onto a hard surface inside an invisible box. The rightmost images show a low viscosity simple fluid and an extremely viscous simple fluid for comparison. The viscoelastic examples have the same viscosity as the low viscosity fluid. For appropriate values of α and γ, the viscoelastic fluid actually bounces.

where α is the material's elastic decay rate, which determines the rate of plastic flow.

Assuming that the initial total and plastic strains are both zero, Equation (2.14) and Equation (2.17) can be combined to compute the time derivative of the elastic strain, which takes into account changes to both the total and plastic strains

$$\dot{\boldsymbol{\epsilon}}^{\text{Elc}} = \frac{\boldsymbol{\nabla}\,\boldsymbol{u} + (\boldsymbol{\nabla}\,\boldsymbol{u})^{\mathsf{T}}}{2} - \alpha \frac{\boldsymbol{\epsilon}'}{\|\boldsymbol{\epsilon}'\|} \max\{0, \|\boldsymbol{\epsilon}'\| - \gamma\} \quad . \tag{2.18}$$

The images of a viscoelastic cube being dropped onto a hard surface are shown in Figure 2.5 to illustrate some effects generated by varying the elastic decay rate α and the plastic yield point γ.

2.4 Advecting Strain

The formulation of the time derivative used in Equation (2.18) does not take into account the movement of the material through space. Like velocity or any other fluid property, the

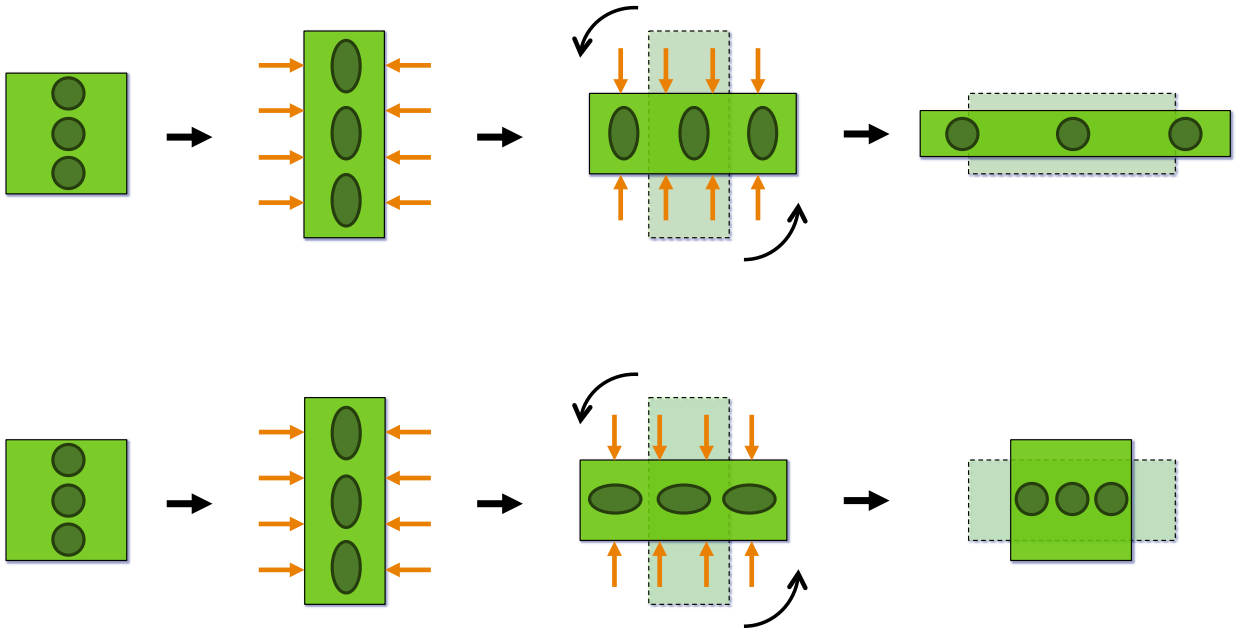

Figure 2.6: A cube of viscoelastic material is compressed in x-direction, rotated around the z-axis and released. The top row shows the effect of advecting the elastic strain *as is*. Since the strain tensors (denoted by green circles) are simply translated during the rotation, they cause the cube stretch to further in x and compress in y upon release. The bottom row shows the effect of both advecting and rotating the strain; the cube recovers its original shape.

elastic strain must be advected with the fluid. To do so, the time rate of change of elastic strain tensor must be expressed as a material derivative. The simplest method is to advect the strain tensor just like any other scalar or vector property,

$$\dot{\boldsymbol{\epsilon}} \equiv \frac{\partial \boldsymbol{\epsilon}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\epsilon} \quad . \tag{2.19}$$

Using this simple material derivative in Equation (2.18) yields linear viscoelasticity

$$\frac{\partial \boldsymbol{\epsilon}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{\epsilon} + \frac{\boldsymbol{\nabla}\,\boldsymbol{u} + (\boldsymbol{\nabla}\,\boldsymbol{u})^{\mathsf{T}}}{2} - \alpha \frac{\boldsymbol{\epsilon}'}{\|\boldsymbol{\epsilon}'\|} \max\left\{0, \|\boldsymbol{\epsilon}'\| - \gamma\right\} \quad . \tag{2.20}$$

The material derivative in Equation (2.19) suggests that the strain tensor is simply translated along with the fluid. Suppose that a cube of viscoelastic material is compressed in the x-direction, rotated 90 degrees around the z-axis, and released (see Figure 2.6). When the cube is first deformed, the strain tensor encodes a compression in the x-direction and a stretch in the y-direction (due to incompressibility). If Equation (2.20) is used to update the strain during the rotation, the components of the strain tensor will simply be translated along the circular path *as is*. Upon release, it will stretch in x and compress in y, causing it to deviate further from its original cube shape. To correct this, we must rotate the strain tensor as we translate it along the path [Bird *et al.*, 1987a; Mao and Yang, 2006; Irving, 2007].

The material derivative that both translates and rotates a tensor along a path is called *Jauman* or *co-rotational* derivative and can be expressed as

$$\dot{\boldsymbol{\epsilon}} \equiv \frac{\partial \boldsymbol{\epsilon}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\epsilon} + \boldsymbol{\epsilon}\boldsymbol{W} - \boldsymbol{W}\boldsymbol{\epsilon} \quad , \tag{2.21}$$

where $\boldsymbol{W} = \left(\boldsymbol{\nabla}\,\boldsymbol{u} - (\boldsymbol{\nabla}\,\boldsymbol{u})^{\mathsf{T}}\right)/2$ is the anti-symmetric spin tensor which computes the rate of rotation in the material[3]. Using the Jauman derivative in Equation (2.18) we obtain our final, rotationally invariant form the strain update equation,

$$\frac{\partial \boldsymbol{\epsilon}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{\epsilon} + \boldsymbol{W}\boldsymbol{\epsilon} - \boldsymbol{\epsilon}\boldsymbol{W} + \frac{\boldsymbol{\nabla}\,\boldsymbol{u} + (\boldsymbol{\nabla}\,\boldsymbol{u})^{\mathsf{T}}}{2} - \alpha \frac{\boldsymbol{\epsilon}'}{\|\boldsymbol{\epsilon}'\|} \max\left\{0, \|\boldsymbol{\epsilon}'\| - \gamma\right\} \ . \tag{2.23}$$

2.5 Imposing Incompressibility

In incompressible flows, the governing equations of the fluid do not provide an explicit equation of state for the pressure. Therefore we cannot directly compute the pressure that would resist volume change in the fluid. Instead, we use the *pressure projection method* introduced by Chorin [1968] and adopted by Stam [1999] to provide a set of stable methods for simulating incompressible fluids. In this method, we first compute an intermediate flow

[3]A tensor $\boldsymbol{T}$ is said to be *invariant under rotation*, if it transforms as $\boldsymbol{T'} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\mathsf{T}}$ under a super-posed rigid-body rotation $\boldsymbol{Q}$. Note that, even if $\boldsymbol{T}$ is rotationally invariant, its time derivative in general is not, since the time derivative transforms as

$$\begin{aligned} \dot{\boldsymbol{T}}' = \dot{\overline{\boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\mathsf{T}}}} &= \dot{\boldsymbol{Q}}\boldsymbol{T}\boldsymbol{Q}^{\mathsf{T}} + \boldsymbol{Q}\dot{\boldsymbol{T}}\boldsymbol{Q}^{\mathsf{T}} + \boldsymbol{Q}\boldsymbol{T}\dot{\boldsymbol{Q}}^{\mathsf{T}} \\ &= \boldsymbol{Q}\dot{\boldsymbol{T}}\boldsymbol{Q}^{\mathsf{T}} + \boldsymbol{W}\boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\mathsf{T}} - \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{W} = \boldsymbol{Q}\dot{\boldsymbol{T}}\boldsymbol{Q}^{\mathsf{T}} + \boldsymbol{W}\boldsymbol{T}' - \boldsymbol{T}'\boldsymbol{W} \\ \dot{\boldsymbol{T}}' - \boldsymbol{W}\boldsymbol{T}' + \boldsymbol{T}'\boldsymbol{W} &= \boldsymbol{Q}\dot{\boldsymbol{T}}\boldsymbol{Q}^{\mathsf{T}} \ , \end{aligned} \tag{2.22}$$

where $\boldsymbol{W} = \dot{\boldsymbol{Q}}\boldsymbol{Q}^{\mathsf{T}} = -\boldsymbol{W}^{\mathsf{T}} = -\boldsymbol{Q}\dot{\boldsymbol{Q}}^{\mathsf{T}}$. In essence the Jauman derivative compensates for the extra terms in the left-hand side of Equation (2.22) and in effect it rotates $\boldsymbol{T}$ using $\boldsymbol{W}$. Therefore the Jauman derivative is invariant under rotation.

field that is not incompressible, and then use the continuity equation Equation (2.11) as a constraint to compute a pressure field that would develop instantaneously to make the flow divergence-free. Finally, we project this pressure field onto the intermediate flow field to obtain a divergence-free velocity field.

The algorithm of the pressure projection method is based on a purely mathematical observation, called *Helmholtz-Hodge decomposition*, which states that every vector field $\boldsymbol{v}^*$ can be decomposed into a divergence-free and a curl-free part

$$\boldsymbol{v}^* = \boldsymbol{v} + \boldsymbol{\nabla}\phi \quad , \tag{2.24}$$

where $\boldsymbol{v}$ is a divergence-free vector field ($\boldsymbol{\nabla} \cdot v = 0$), ϕ is a scalar field and $\boldsymbol{\nabla}\phi$ is a curl-free vector field ($\boldsymbol{\nabla} \times \boldsymbol{\nabla}\phi = 0$).

If we apply the divergence operator on both sides of Equation (2.24) and rearrange the result, we obtain the Poisson equation for ϕ

$$\boldsymbol{\nabla}^2\phi = \boldsymbol{\nabla} \cdot \boldsymbol{v}^* \quad . \tag{2.25}$$

Once we solve Equation (2.25) for the scalar field ϕ, we can project $\boldsymbol{\nabla}\phi$ on $\boldsymbol{v}^*$ to obtain the divergence-free $\boldsymbol{v}$ using Equation (2.24)

$$\boldsymbol{v} = \boldsymbol{v}^* - \boldsymbol{\nabla}\phi \quad . \tag{2.26}$$

We can use the above procedure to project $\boldsymbol{\nabla}p$ onto some intermediate non-divergence-free velocity field $\boldsymbol{u}^*$ to obtain a final incompressible velocity field $\boldsymbol{u}$. Assume we compute all terms of Equation (2.10), except for the pressure forces as an intermediate velocity field $\boldsymbol{u}^*$

$$\frac{\partial \boldsymbol{u}^*}{\partial t} = -\left(\boldsymbol{u} \cdot \boldsymbol{\nabla}\right)\boldsymbol{u} + \frac{\mu_v}{\rho}\boldsymbol{\nabla}^2\boldsymbol{u} + \frac{\mu_e}{\rho}\boldsymbol{\nabla} \cdot \boldsymbol{\epsilon} + \frac{\boldsymbol{f}}{\rho} \quad . \tag{2.27}$$

Then we can rearrange Equation (2.10) as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\boldsymbol{\nabla}p}{\rho} = \frac{\partial \boldsymbol{u}^*}{\partial t} \quad . \tag{2.28}$$

Equation (2.28) has the same form of Equation (2.24) where $\boldsymbol{v}^* = \partial \boldsymbol{u}^*/\partial t$, $\boldsymbol{v} = \partial \boldsymbol{u}/\partial t$ and $\phi = p/\rho$. Therefore we can compute p by solving the Poisson equation for pressure

$$\frac{1}{\rho}\boldsymbol{\nabla}^2 p = \boldsymbol{\nabla} \cdot \frac{\partial \boldsymbol{u}^*}{\partial t} \quad , \tag{2.29}$$

and project $\boldsymbol{\nabla}p/\rho$ on $\partial \boldsymbol{u}^*/\partial t$, and then integrate the result to obtain the final divergence-free velocity field $\boldsymbol{u}$

$$\boldsymbol{u} = \boldsymbol{u}^* - \int \frac{1}{\rho}\boldsymbol{\nabla}p \; dt \quad . \tag{2.30}$$

2.6 Boundary Conditions

To solve our system of equations, we must satisfy a set of boundary conditions for fluid velocity $\boldsymbol{u}$, fluid pressure p and elastic strain $\boldsymbol{\epsilon}$. In this thesis we consider two types of boundaries: *solid boundaries* such as walls or moving collision objects, and the *free surface boundary*. At the solid boundaries, to prevent fluid entering into the colliders, the normal component of the fluid velocity must match the normal component of velocity of the solid boundary

$$\boldsymbol{u} \cdot \boldsymbol{n} = \bar{\boldsymbol{u}} \cdot \boldsymbol{n} \quad , \tag{2.31}$$

where $\boldsymbol{n}$ is the normal of the solid boundary and $\bar{\boldsymbol{u}}$ is the velocity of the solid collider boundary ($\bar{\boldsymbol{u}} = 0$ for a static wall). For the tangential component of the fluid velocity we have two options. We can either leave it unspecified to let it flow freely on the solid surface, called *slip boundary condition*, or we can set it to the tangential component of the solid boundary velocity to make the fluid stick to the collider, called *no-slip boundary condition*

$$\boldsymbol{u} \cdot \boldsymbol{t} = \bar{\boldsymbol{u}} \cdot \boldsymbol{t} \quad , \tag{2.32}$$

where $\boldsymbol{t}$ is the tangent of the solid boundary. We also need to make sure that neither the pressure gradient $\boldsymbol{\nabla} p$, by pressure projection, nor the elastic strain $\boldsymbol{\epsilon}$, by adding elastic forces, cause any additional acceleration normal to the solid boundaries, i.e.

$$\boldsymbol{\nabla} p \cdot \boldsymbol{n} = \frac{\partial p}{\partial \boldsymbol{n}} = 0 \quad , \tag{2.33}$$

and

$$(\boldsymbol{\nabla} \cdot \boldsymbol{\epsilon}) \cdot \boldsymbol{n} = 0 \quad . \tag{2.34}$$

For free surface boundaries both velocity and elastic strain must be constant across the boundary, i.e. their rate of change normal to the free surface boundary should be zero

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = 0 \quad , \tag{2.35}$$

and

$$\frac{\partial \boldsymbol{\epsilon}}{\partial \boldsymbol{n}} = 0 \quad . \tag{2.36}$$

Since we do not simulate the air on the other side of the free-surface we assume that the pressure at the free surface equals the air pressure which is negligible (i.e. zero) compared to water pressure

$$p = 0 \quad . \tag{2.37}$$

2.7 Putting it all Together

To summarize, the governing equation of our model is the modified Navier–Stokes in Equation (2.10), where we add extra elastic forces depending on elastic strain. We impose incompressibility using the pressure projection method, where we first solve for pressure in Equation (2.29), and then we project the pressure to obtain the final divergence free velocity using Equation (2.30). Finally we update the elastic strain using Equation (2.23), where we advect the strain trough the fluid and decay it, if it exceeds the plastic yield point. All these steps are subject to the velocity, strain and pressure boundary conditions discussed in Section 2.6. The details of the numerical methods solving these equations using Eulerian grid based techniques are discussed in Chapter 3.

Chapter 3

Simulation

The Eulerian framework we use for fluid simulation is based on the method described in [Enright *et al.*, 2002]. This framework consists of three primary components that work together to simulate incompressible fluids with free surfaces. The first is a rectilinear grid that stores the pressure and velocity values that define the fluid's state. These pressure and velocity values are stored in a staggered fashion using the MAC grid scheme introduced by Harlow and Welch [1965]. Second the Navier–Stokes equation is discretized on this grid and solved term-by-term using the operator splitting method described in [Stam, 1999]. The third component is a function whose level-set at zero locates the boundaries of the fluid. The function is represented by a combination of particles and signed surface distance values defined on a second rectilinear grid. The free surface of the fluid is tracked by evolving the particles and the distance values based on the motion of the fluid. Another alternative for tracking the free surface is the semi-Lagrangian contouring method introduced by Bargteil et al. [2006]. The semi-Lagrangian contouring method uses an explicit mesh instead of a dense set of particles to compute and evolve the distance values, and has the additional advantage of robustly tracking surface properties through the simulation. For completeness, in this thesis we provide details on a simple implementation of the system described in [Enright *et al.*, 2002] which is summarized in the middle blue box in Figure 3.1. For a more detailed description of this simulation methodology we refer the reader to Bridson's excellent summary on fluid simulation for computer graphics [Bridson, 2008].

To augment the fluid simulator described above with viscoelastic behavior, we first discretize the elastic strain tensor on the computational grid. We extend the MAC grid scheme for staggering tensors on the grid. Then at every time-step n we compute the elastic forces due to elastic strain, i.e $\boldsymbol{f}_e^n = \mu_e \boldsymbol{\nabla} \cdot \boldsymbol{\epsilon}^n$, and pass the elastic forces to the fluid simulator as additional body forces as show in Equation (2.10). Finally at the end of the time-step the elastic strain is updated by solving Equation (2.23) using the new divergence-free velocity field $\boldsymbol{u}^{n+1}$ we obtained from the fluid simulator. These two additions are shown in orange boxes in Figure 3.1. Next we discuss the details of our staggered grid scheme and provide details on each step of our algorithm in Figure 3.1 for simulating viscoelastic fluids.

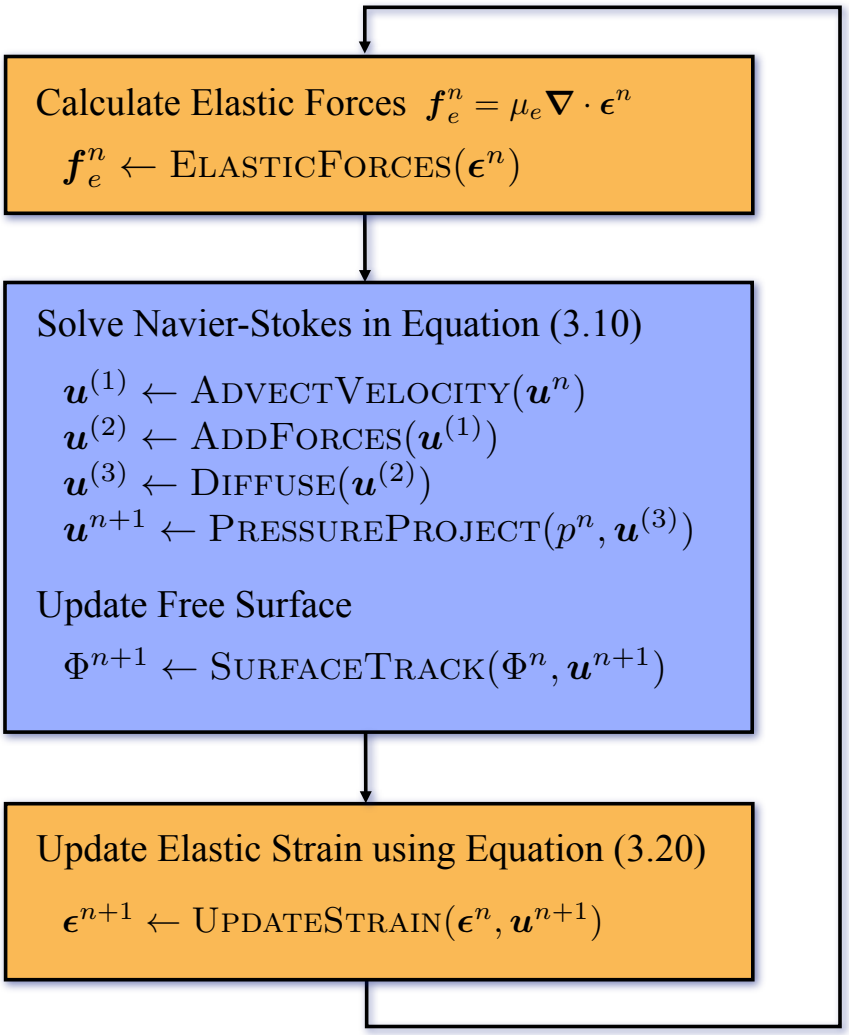

Figure 3.1: Algorithm for each time-step n of our method: We first compute elastic forces and pass them to an Eulerian based fluid simulator. The fluid simulator solves the Navier–Stokes equations using the elastic forces as additional body forces, and tracks the surface. We then use the newly computed incompressible velocity to update the elastic strain tensor.

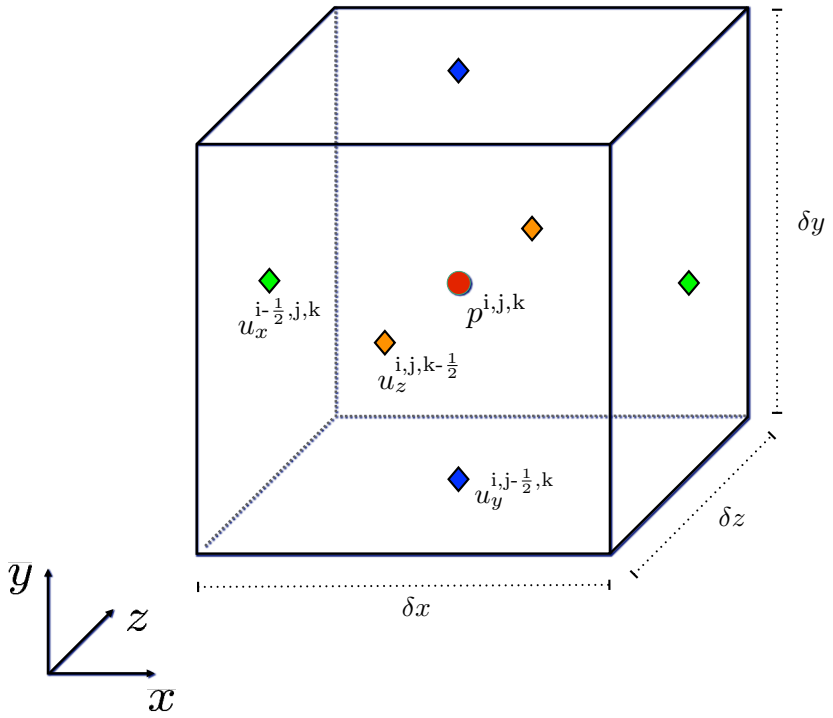

Figure 3.2: The staggered MAC grid scheme: The rectilinear grid is divided into cells which have a dimension of δx, δy and δz in x, y and z directions, respectively. The location of each cell center is identified by an integer triple (i,j,k). The scalar pressure p is stored at the cell center. The components of the velocity vector $\boldsymbol{u}$ are staggered where each component is stored at the corresponding face center. The superscript on each value denotes its location on the grid.

3.1 Choosing a Grid

To simulate viscoelastic fluids using the algorithm outlined in Figure 3.1, we first discretize our computational domain into a rectilinear grid. We store the scalar pressure, velocity vector and strain tensor values that define the fluid's state on this grid, and compute their values by numerically solving Equations (2.10), (2.29), (2.30) and (2.23) using the symmetrical, unbiased, second order accurate central finite differences. To avoid the instability problems caused by the decoupling of the scalar pressure and velocity vector fields that plague the collocated schemes, we use the staggered MAC grid scheme originally described by Harlow and Welch [1965]. In this scheme the pressure p and the components of the velocity vector $\boldsymbol{u}$ are stored in a staggered fashion on the grid as show in Figure 3.2. The key observation in this scheme is that when we discretize the gradient of a scalar field (e.g. $\boldsymbol{\nabla} p$), the components of the resulting vector field are computed exactly where the components of the velocity are

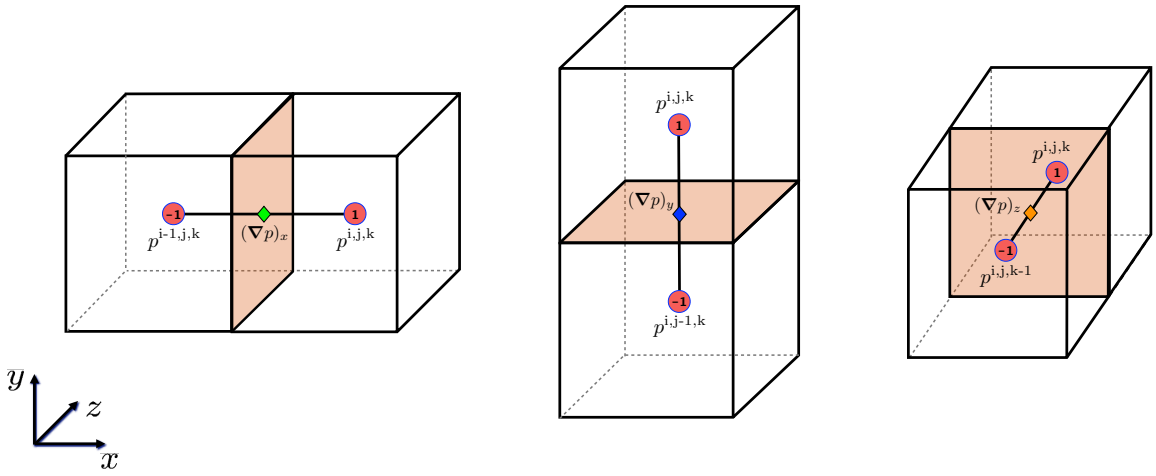

Figure 3.3: Components of ∇p are computed on the grid using central finite differences. Each component is computed on the corresponding face center.

stored. Similarly when we discretize the divergence of a vector field (e.g. $\nabla \cdot \boldsymbol{u}$) the resulting scalar field is computed exactly where the scalar pressure values are stored. Therefore the staggered MAC grid scheme provides a strong coupling between all pressure and velocity components when we solve Equations (2.29) and (2.30). To demonstrate this in more detail, using the central differences, we can discretize ∇p as

$$\begin{aligned}
(\nabla p)_x^{\text{i-}\frac{1}{2}\text{,j,k}} &= \frac{\left(p^{\text{i,j,k}} - p^{\text{i-1,j,k}}\right)}{\delta x} \\
(\nabla p)_y^{\text{i,j-}\frac{1}{2}\text{,k}} &= \frac{\left(p^{\text{i,j,k}} - p^{\text{i,j-1,k}}\right)}{\delta y} \\
(\nabla p)_z^{\text{i,j,k-}\frac{1}{2}} &= \frac{\left(p^{\text{i,j,k}} - p^{\text{i,j,k-1}}\right)}{\delta z}
\end{aligned} \tag{3.1}$$

Note that the components of ∇p are computed at the midpoints between two cell centers which correspond to the center of the faces between them, exactly where the velocity components are defined as shown in Figure 3.3. Similarly divergence of the velocity vector $\nabla \cdot \boldsymbol{u}$ can be discretized as

$$(\nabla \cdot \boldsymbol{u})^{\text{i,j,k}} = \frac{u_x^{\text{i+}\frac{1}{2}\text{,j,k}} - u_x^{\text{i-}\frac{1}{2}\text{,j,k}}}{\delta x} + \frac{u_y^{\text{i,j+}\frac{1}{2}\text{,k}} - u_y^{\text{i,j-}\frac{1}{2}\text{,k}}}{\delta y} + \frac{u_z^{\text{i,j,k+}\frac{1}{2}} - u_z^{\text{i,j,k-}\frac{1}{2}}}{\delta z} \quad . \tag{3.2}$$

Note the scalar value $\nabla \cdot \boldsymbol{u}$ is computed at the midpoint between the velocity components at face centers which correspond to the center of the cell where the scalar pressure values

are stored as shown in Figure 3.4.

In addition to the scalar pressure and the velocity vector fields, we need to extend the classic MAC grid scheme to store the strain tensor field on the grid. In Equation (2.23) we use the tensor field $\boldsymbol{\nabla u}$ to update the elastic strain. Using the observation above, to create a strong coupling between velocity and strain tensor, components of the elastic strain tensor need to be defined exactly where the components of the $\boldsymbol{\nabla u}$ are computed. Using the central finite differences, we discretize $\boldsymbol{\nabla u}$ on the grid as

$$
\begin{aligned}
(\boldsymbol{\nabla u})^{\mathrm{i,j,k}}_{xx} &= \frac{u_x^{\mathrm{i+\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j,k}}}{\delta x} \\
(\boldsymbol{\nabla u})^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}}_{xy} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j-1,k}}}{\delta y} \\
(\boldsymbol{\nabla u})^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}}_{xz} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j,k-1}}}{\delta z} \\
(\boldsymbol{\nabla u})^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}}_{yx} &= \frac{u_y^{\mathrm{i,j-\frac{1}{2},k}} - u_y^{\mathrm{i-1,j-\frac{1}{2},k}}}{\delta x} \\
(\boldsymbol{\nabla u})^{\mathrm{i,j,k}}_{yy} &= \frac{u_y^{\mathrm{i,j+\frac{1}{2},k}} - u_y^{\mathrm{i,j-\frac{1}{2},k}}}{\delta y} \\
(\boldsymbol{\nabla u})^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}}_{yz} &= \frac{u_y^{\mathrm{i,j-\frac{1}{2},k}} - u_y^{\mathrm{i,j-\frac{1}{2},k-1}}}{\delta z} \\
(\boldsymbol{\nabla u})^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}}_{zx} &= \frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}} - u_z^{\mathrm{i-1,j,k-\frac{1}{2}}}}{\delta x} \\
(\boldsymbol{\nabla u})^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}}_{zy} &= \frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}} - u_z^{\mathrm{i,j-1,k-\frac{1}{2}}}}{\delta y} \\
(\boldsymbol{\nabla u})^{\mathrm{i,j,k}}_{zz} &= \frac{u_z^{\mathrm{i,j,k+\frac{1}{2}}} - u_z^{\mathrm{i,j,k-\frac{1}{2}}}}{\delta z}
\end{aligned}
\tag{3.3}
$$

Note that the diagonal components of $\boldsymbol{\nabla u}$ are located at cell centers and off-diagonal components are located on the corresponding edge centers (see Figure 3.5). This leads to the final staggered tensor scheme for storing the symmetric elastic strain shown in Figure 3.6 which is a generalization of the 2D scheme described in [Gerritsma, 1996].

In addition to the computational grid described above, we also maintain a separate grid to store the scalar signed distance Φ values we use to track the surface as discussed in Section 3.4. This grid has a higher resolution and is staggered with respect to the computational grid, so that every cell center, face center, edge center and corner on the computational domain is a cell center in the Φ grid and thus has a signed distance value associated with it as shown in Figure 3.9. For each pressure, velocity and strain tensor component, we use its corresponding Φ value to decide if it is free to be computed by the simulator or if it needs to

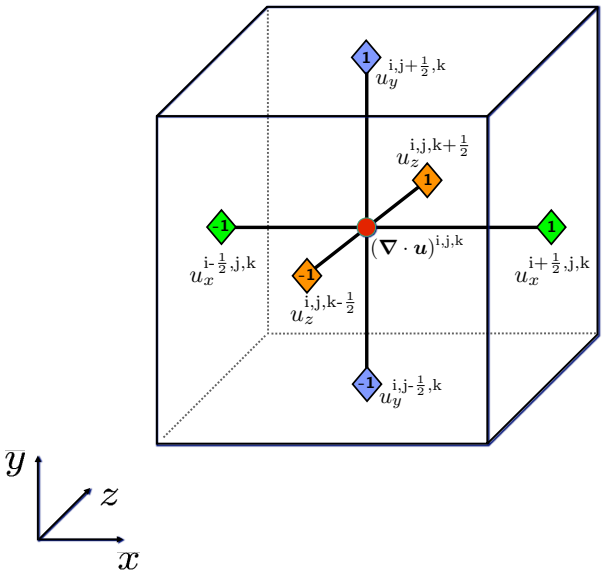

Figure 3.4: The scalar value $\nabla \cdot \boldsymbol{u}$ is computed at the cell center on the grid using central finite differences.

be set as a boundary condition. Every pressure, velocity and strain component which has a negative Φ value, i.e inside the fluid, is labeled as *free*. All the others are labeled *non-free* and are used to set the boundary conditions as discussed in Section 3.6.

3.2 Computing Elastic Forces

The first step of our method for simulating viscoelastic fluids is to compute the elastic forces $\boldsymbol{f}_e$ due to elastic strain $\boldsymbol{\epsilon}$ for the current time-step n as

$$\boldsymbol{f}_e^n = \mu_e \nabla \cdot \boldsymbol{\epsilon}^n \tag{3.4}$$

Using the central finite differences, we can discretize Equation (3.4) on the staggered grid as

$$\begin{aligned}
f_{e_x}{}^{i-\frac{1}{2},j,k} &= \mu_e\left(\frac{\epsilon_{xx}^{i,j,k} - \epsilon_{xx}^{i-1,j,k}}{\delta x} + \frac{\epsilon_{xy}^{i-\frac{1}{2},j+\frac{1}{2},k} - \epsilon_{xy}^{i-\frac{1}{2},j-\frac{1}{2},k}}{\delta y} + \frac{\epsilon_{xz}^{i-\frac{1}{2},j,k+\frac{1}{2}} - \epsilon_{xz}^{i-\frac{1}{2},j,k-\frac{1}{2}}}{\delta z}\right) \\
f_{e_y}{}^{i,j-\frac{1}{2},k} &= \mu_e\left(\frac{\epsilon_{xy}^{i+\frac{1}{2},j-\frac{1}{2},k} - \epsilon_{xy}^{i-\frac{1}{2},j-\frac{1}{2},k}}{\delta x} + \frac{\epsilon_{yy}^{i,j,k} - \epsilon_{yy}^{i,j-1,k}}{\delta y} + \frac{\epsilon_{yz}^{i,j-\frac{1}{2},k+\frac{1}{2}} - \epsilon_{yz}^{i,j-\frac{1}{2},k-\frac{1}{2}}}{\delta z}\right) \\
f_{e_z}{}^{i,j,k-\frac{1}{2}} &= \mu_e\left(\frac{\epsilon_{xz}^{i+\frac{1}{2},j,k-\frac{1}{2}} - \epsilon_{xz}^{i-\frac{1}{2},j,k-\frac{1}{2}}}{\delta x} + \frac{\epsilon_{yz}^{i,j+\frac{1}{2},k-\frac{1}{2}} - \epsilon_{yz}^{i,j-\frac{1}{2},k-\frac{1}{2}}}{\delta y} + \frac{\epsilon_{zz}^{i,j,k} - \epsilon_{zz}^{i,j,k-1}}{\delta z}\right)
\end{aligned} \tag{3.5}$$

Note that due to the staggering of the elastic strain, the components of the vector field $\nabla \cdot \boldsymbol{\epsilon}$ are computed on the face centers, exactly where the velocity components are defined as shown

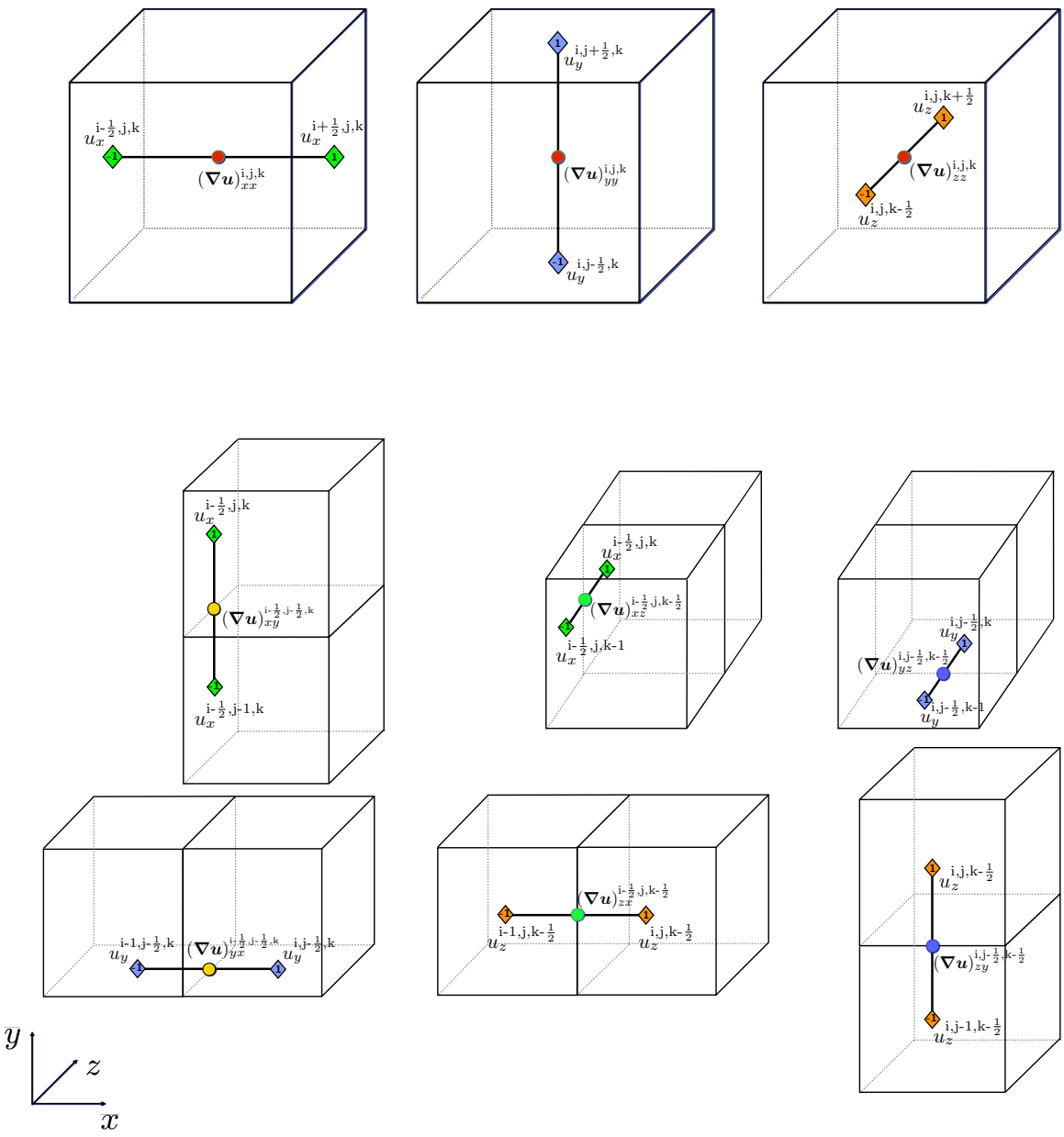

Figure 3.5: Components of the tensor $\nabla \boldsymbol{u}$ computed on the grid using central finite differences. Diagonal components are computed at the cell centers (top row) and the off-diagonal components are computed on edge centers (bottom rows).

in Figure 3.7. Therefore the acceleration due to the elastic forces can be integrated directly into the velocity field without the need of extra interpolation. The details of computing the elastic forces on the entire grid are provided in Algorithm 1.

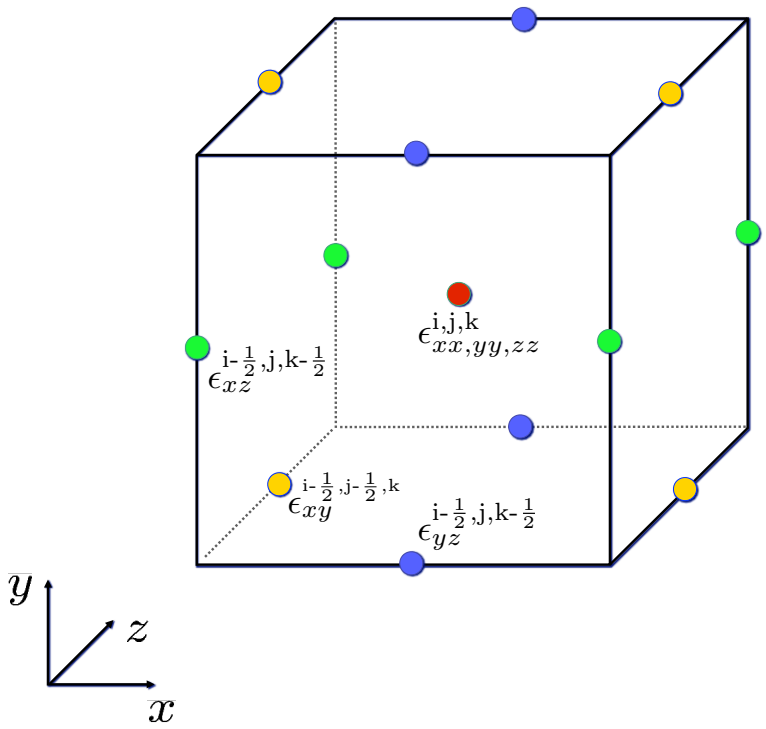

Figure 3.6: Components of the elastic strain tensor $\boldsymbol{\epsilon}$ are staggered on the grid. The diagonal entries ϵ_{xx}, ϵ_{yy} and ϵ_{zz} are located at cell centers whereas off-diagonal components ϵ_{xy}, ϵ_{xz} and ϵ_{yz} are located at the centers of the edges perpendicular to the component directions (note that since $\boldsymbol{\epsilon}$ is symmetric, components ϵ_{yx}, ϵ_{zx} and ϵ_{zy} do not need to be stored explicitly on the grid). The superscript in each component of $\boldsymbol{\epsilon}$ denotes the location of the component on the grid.

3.3 Solving Navier–Stokes

The Navier–Stokes equation in Equation (2.10) is composed of multiple terms that represent different types of PDEs. Since each one of these terms have different requirements, there is no single numerical method that works best for all of them. For example, the nonlinear advection term can become unstable under direct solution methods and can be best solved by a method of characteristics such as the semi-Lagrangian advection. Additionally, while it would be stable enough to time integrate the gravity and the elastic force terms using an explicit integration scheme, the stiff viscous forces are best solved by an implicit time integration scheme. Finally, to impose the incompressibility condition in Equation (2.11), we first need to compute a pressure field and then project the pressure gradient onto an intermediate velocity field to obtain the final divergence-free velocity. Therefore, instead of solving Equation (2.10) directly as a whole, we employ the *operator splitting method*

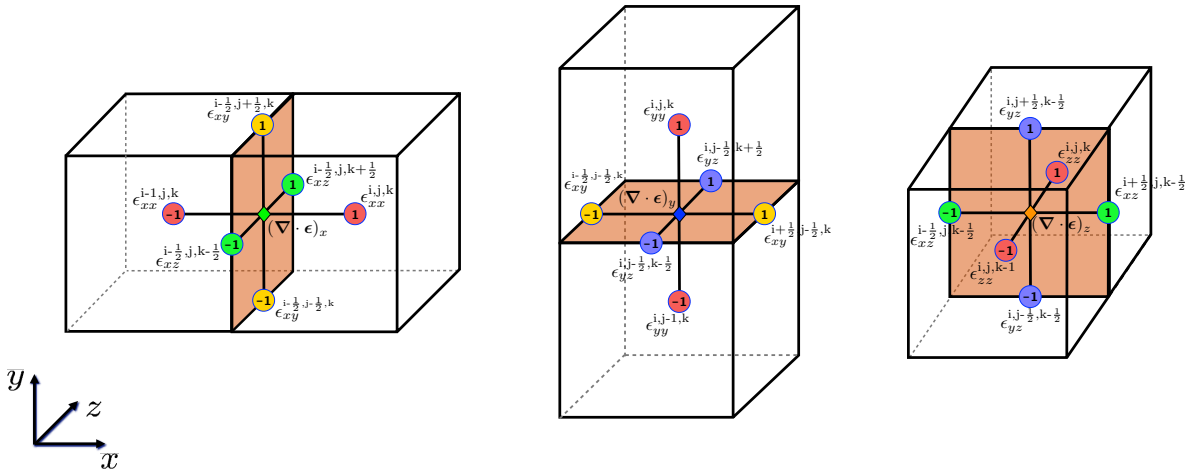

Figure 3.7: The stencils used to compute the x, y and z components of the divergence of a staggered tensor field on the grid. Resulting is a vector field defined on the face centers exactly where the velocity components are defined.

introduced by Stam [1999] where we split Equation (2.10) and solve each term separately using the best method suited for that term.

At each time-step n, we first advect the velocity $\boldsymbol{u}^n$ to obtain an intermediate velocity $\boldsymbol{u}^{(1)}$

$$\frac{\partial \boldsymbol{u}^{(1)}}{\partial t} = -\left(\boldsymbol{u}^n \cdot \boldsymbol{\nabla}\right) \boldsymbol{u}^n \quad . \tag{3.6}$$

Second we add the user defined body forces and the elastic forces to obtain another intermediate velocity $\boldsymbol{u}^{(2)}$

$$\frac{\partial \boldsymbol{u}^{(2)}}{\partial t} = \frac{1}{\rho}\left(\boldsymbol{f} + \boldsymbol{f_e}\right) \tag{3.7}$$

Next we integrate viscous forces into $\boldsymbol{u}^{(2)}$ to obtain our next intermediate velocity field $\boldsymbol{u}^{(2)}$

$$\frac{\partial \boldsymbol{u}^{(3)}}{\partial t} = \frac{\mu_v}{\rho}\boldsymbol{\nabla}^2 \boldsymbol{u}^{(2)} \tag{3.8}$$

Since $\boldsymbol{u}^{(3)}$ is not guaranteed to be divergence-free, we solve for pressure p^n

$$\boldsymbol{\nabla}^2 p^n = \rho \boldsymbol{\nabla} \cdot \frac{\partial \boldsymbol{u}^{(3)}}{\partial t} \quad , \tag{3.9}$$

and project it on the intermediate velocity field $\boldsymbol{u}^{(3)}$ to obtain the final divergence-free velocity $\boldsymbol{u}^{n+1}$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{(3)} - \int \frac{1}{\rho} \boldsymbol{\nabla} p \, dt \quad . \tag{3.10}$$

The top level algorithm for solving the Navier–Stokes equations using operator splitting is shown in Figure 3.1. The details of the methods we use for solving each step outlined above are discussed below.

3.3.1 Advecting Velocity

One of the first methods developed for solving the nonlinear advection term in Equation (3.6) is the direct solution method described in [Harlow and Welch, 1965] which was adopted by Foster et al. [1996] to simulate fluids for computer graphics. In this method, Equation (3.6) is discretized on the grid using central finite differences, and integrated in time using the forward Euler method. However, since every grid point only uses its immediate neighbors to compute the properties moving through the grid, any property value that moves more than the size of its compact neighborhood (a single cell size, i.e. $\max \|\delta x, \delta y, \delta z\|$) in a given time-step δt, cannot be resolved by this method. This imposes a rather strict limit on the time-step, such that

$$\delta t < \frac{\max \|\delta x, \delta y, \delta z\|}{\boldsymbol{u}_{max}} \quad , \tag{3.11}$$

where $\boldsymbol{u}_{max}$ is the maximum velocity anywhere in the fluid. Moreover, it can be shown that using central finite differences and forward Euler method to solve the advection equation is unconditionally unstable, no matter how small the time step is (the above limit only assures that the method is accurate within a given time-step) [Bridson, 2008].

To avoid these limitations we use the unconditionally stable semi-Lagrangian advection method introduced by Stam [Stam, 1999]. The fundamental idea behind this method is the fact that in an incompressible fluid each particle traces a unique path, called a *characteristic*. In a Lagrangian formulation, solving the advection equation is as simple as moving each particle along its characteristics. However, in an Eulerian formulation we are trying to compute values of properties we are interested in at fixed points on the grid. Since each particle traces a unique path, at any point in time for any point on the grid there will be a unique particle that will end up at that point. Therefore, if we can find that particle by going backwards in time, we can find the value of any property we are interested in. We can use this procedure to advect any property, including the velocity. For example, to advect the x-component of the velocity we start from the center of an x-face $\boldsymbol{x}^{\mathrm{i}-\frac{1}{2},\mathrm{j},\mathrm{k}}$, trace a path back to a point $\boldsymbol{x_f}$ and use the x-component of the velocity at $\boldsymbol{x}_f$, i.e. $\boldsymbol{u}_x(\boldsymbol{x}_f)$ as the new x-component of the velocity at the grid point $\boldsymbol{x}^{\mathrm{i}-\frac{1}{2},\mathrm{j},\mathrm{k}}$ (see Figure 3.8). To trace the path of

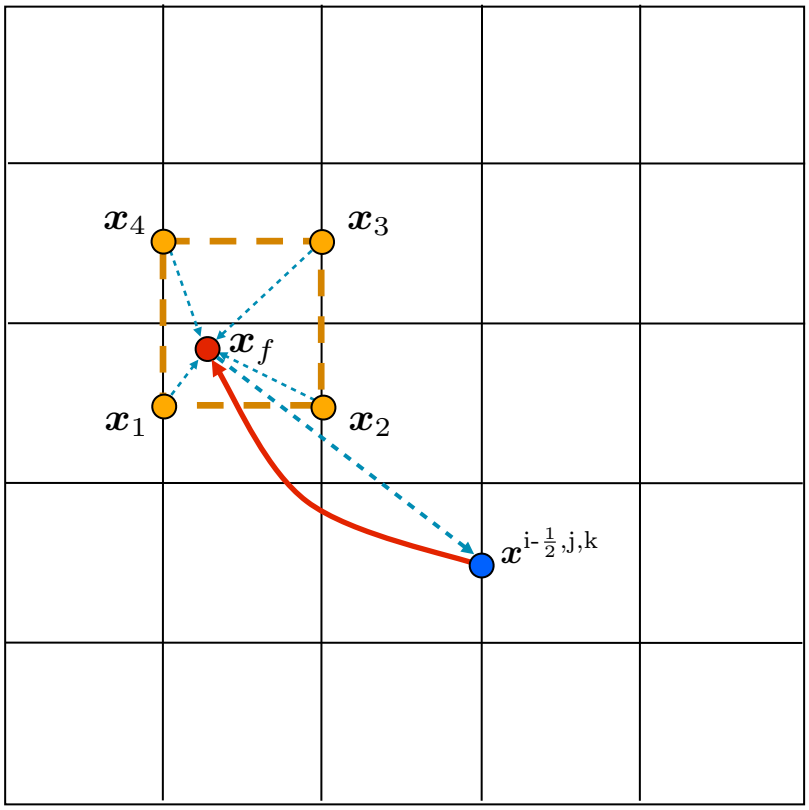

Figure 3.8: Basic steps of the semi-Lagrangian algorithm for advecting the x-component of the velocity (shown in 2D for simplicity). Starting from an x-face center $\boldsymbol{x}^{\text{i-}\frac{1}{2},\text{j,k}}$, we trace back to a point $\boldsymbol{x}_f$. We interpolate the x-component of the velocity at $\boldsymbol{x}_f$ using the x-component of the velocities at $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ and $\boldsymbol{x}_4$, and use it to set the new x-component of the velocity at $\boldsymbol{x}^{\text{i-}\frac{1}{2},\text{j,k}}$.

a particle we use the second order Runge-Kutta time integration method

$$\begin{aligned} \boldsymbol{x}_{mid} &= \boldsymbol{x}^{\text{i-}\frac{1}{2},\text{j,k}} - \frac{\delta t}{2}\, \boldsymbol{u}(\boldsymbol{x}^{\text{i-}\frac{1}{2},\text{j,k}}) \\ \boldsymbol{x}_f &= \boldsymbol{x}^{\text{i-}\frac{1}{2},\text{j,k}} - \delta t\, \boldsymbol{u}(\boldsymbol{x}_{mid}) \quad . \end{aligned} \tag{3.12}$$

Once we find the position $\boldsymbol{x}_f$ of the particle, we compute its velocity $\boldsymbol{u}(\boldsymbol{x}_f)$ using tri-linear interpolation on the grid. Finally, we use x-component of this new velocity as the x-component of the final advected velocity

$$u_x^{(1)}(\boldsymbol{x}_c) = u_x(\boldsymbol{x}_f) \quad . \tag{3.13}$$

We repeat this process to compute every component of $\boldsymbol{u}^{(1)}$. The final algorithm for advecting the velocity is shown in Algorithm 2.

3.3.2 Adding Forces

In the next step we integrate the user defined body forces and the elastic forces we computed in Section 3.2. In our implementation, we only consider gravity forces as body forces $\boldsymbol{f} = \rho \boldsymbol{g}$, where $\boldsymbol{g}$ is the gravity vector. Using explicit time integration for both gravity and elastic forces, we get

$$\boldsymbol{u}^{(2)} = \boldsymbol{u}^{(1)} + \delta t \left(\boldsymbol{g} + \frac{1}{\rho} \boldsymbol{f_e} \right) \quad . \tag{3.14}$$

Discretizing Equation (3.14) on the grid we obtain

$$\begin{aligned}
({u_x}^{(2)})^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}} &= ({u_x}^{(1)})^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}} + \delta t \left(g_x + \frac{1}{\rho} {f_e}_x{}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}} \right) \\
({u_y}^{(2)})^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}} &= ({u_y}^{(1)})^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}} + \delta t \left(g_y + \frac{1}{\rho} {f_e}_y{}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}} \right) \\
({u_z}^{(2)})^{\mathrm{i},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}} &= ({u_z}^{(1)})^{\mathrm{i},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}} + \delta t \left(g_z + \frac{1}{\rho} {f_e}_z{}^{\mathrm{i},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}} \right)
\end{aligned} \tag{3.15}$$

The details of adding the gravity and elastic forces on the entire grid is provided in Algorithm 3. We should note that, as reported by Irving [2007], using the simple explicit scheme above for integrating elastic forces poses the following restriction on the time-step δt

$$\delta t < \delta x \sqrt{\frac{\rho}{\mu_e}} \quad . \tag{3.16}$$

Therefore using our method for simulating very stiff materials would require small time-steps. Note that we are interested in simulating the fluid end of the spectrum where either μ_e is low and/or the elastic strain is decaying exponentially over time. Therefore in practice, the use of explicit integration does not force us to take small time-steps.

3.3.3 Viscosity

Next we compute the viscous forces which act as internal friction to shear flow. In effect, viscosity in a fluid causes the velocity to dissipate and is formulated by a diffusion equation

$$\frac{\partial \boldsymbol{u}}{\delta t} = \frac{\mu_v}{\rho} \boldsymbol{\nabla}^2 \boldsymbol{u} \quad . \tag{3.17}$$

For fluids with high viscosity the viscous forces can be fairly stiff. This makes the use of explicit time integration impractical. Therefore, we use the implicit method described in

Stam [1999] to solve Equation (3.17). Using the implicit backward Euler time integration scheme, integration of the viscous forces can be formulated as

$$\frac{\boldsymbol{u}^{(3)} - \boldsymbol{u}^{(2)}}{\delta t} = \frac{\mu_v}{\rho}\boldsymbol{\nabla}^2\boldsymbol{u}^{(3)} \quad , \tag{3.18}$$

which can be rearranged as

$$\left(\boldsymbol{I} - \delta t\frac{\mu_v}{\rho}\boldsymbol{\nabla}^2\right)\boldsymbol{u}^{(3)} = \boldsymbol{u}^{(2)} \quad . \tag{3.19}$$

In order to solve Equation (3.19) we need to discretize the Laplacian operator $\boldsymbol{\nabla}^2 \equiv \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}$ on the grid. Applying the central finite differences on each component of the vector $\boldsymbol{\nabla}^2\boldsymbol{u}$, we obtain

$$\begin{aligned}
\left(\boldsymbol{\nabla}^2\boldsymbol{u}\right)_x^{\text{i-}\frac{1}{2}\text{,j,k}} =& \frac{u_x^{\text{i-}\frac{3}{2}\text{,j,k}} + u_x^{\text{i+}\frac{1}{2}\text{,j,k}}}{\delta x^2} + \frac{u_x^{\text{i-}\frac{1}{2}\text{,j-1,k}} + u_x^{\text{i-}\frac{1}{2}\text{,j+1,k}}}{\delta y^2} + \frac{u_x^{\text{i-}\frac{1}{2}\text{,j,k-1}} + u_x^{\text{i-}\frac{1}{2}\text{,j,k+1}}}{\delta z^2} \\
& - 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \\
\left(\boldsymbol{\nabla}^2\boldsymbol{u}\right)_y^{\text{i,j-}\frac{1}{2}\text{,k}} =& \frac{u_y^{\text{i-1,j-}\frac{1}{2}\text{,k}} + u_y^{\text{i+1,j-}\frac{1}{2}\text{,k}}}{\delta x^2} + \frac{u_y^{\text{i,j-}\frac{3}{2}\text{,k}} + u_y^{\text{i,j+}\frac{1}{2}\text{,k}}}{\delta y^2} + \frac{u_y^{\text{i,j-}\frac{1}{2}\text{,k-1}} + u_y^{\text{i,j-}\frac{1}{2}\text{,k+1}}}{\delta z^2} \\
& - 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \\
\left(\boldsymbol{\nabla}^2\boldsymbol{u}\right)_z^{\text{i,j,k-}\frac{1}{2}} =& \frac{u_z^{\text{i-1,j,k-}\frac{1}{2}} + u_z^{\text{i+1,j,k-}\frac{1}{2}}}{\delta x^2} + \frac{u_z^{\text{i,j-1,k-}\frac{1}{2}} + u_z^{\text{i,j+1,k-}\frac{1}{2}}}{\delta y^2} + \frac{u_z^{\text{i,j,k-}\frac{3}{2}} + u_z^{\text{i,j,k+}\frac{1}{2}}}{\delta z^2} \\
& - 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_z^{\text{i,j,k-}\frac{1}{2}}
\end{aligned} \tag{3.20}$$

Using the discretization in Equation (3.20), Equation (3.19) can be expressed as a set of three linear systems, one for each component of $\boldsymbol{u}^{(3)}$

$$\begin{aligned}
K_v {u_x}^{(3)} &= {u_x}^{(2)} \\
K_v {u_y}^{(3)} &= {u_y}^{(2)} \\
K_v {u_z}^{(3)} &= {u_z}^{(2)}
\end{aligned} \tag{3.21}$$

where $K_v = \left(\boldsymbol{I} - \delta t\frac{\mu_v}{\rho}\boldsymbol{\nabla}^2\right)$. We solve each of these systems separately using the conjugate gradient method best described in [Shewchuk, 1994]. However, instead of assembling K_v as a separate matrix, we perform the matrix multiplication on the grid using a set of special functions detailed in Algorithm 5. Instead of K_v, we use these multiplication functions in our implementation of the conjugate gradient algorithm (see Algorithm 4). The final details of integrating viscous forces are provided in Algorithm 6.

3.3.4 Pressure Projection

None of the terms we solved so far operated under the constraint that the velocity field we computed must be divergence-free. Therefore, even though we started with a divergence-free velocity field $\boldsymbol{u}^n$ at the beginning of the time-step n, the last velocity field we computed $\boldsymbol{u}^{(3)}$ is not necessarily incompressible. We need to solve the pressure Poisson equation in Equation (2.29), to obtain the pressure that would make the flow incompressible, and then, using Equation (2.26) we must project the gradient of that pressure onto $\boldsymbol{u}^{(3)}$ to obtain our final divergence-free velocity field $\boldsymbol{u}^{n+1}$.

Integrating Equation (2.29) in time and rearranging it, we obtain

$$\boldsymbol{\nabla}^2 p^n = \frac{\rho}{\delta t}\boldsymbol{\nabla}\cdot\boldsymbol{u}^{(3)} \quad . \tag{3.22}$$

Using the centered finite differences we can discretize $\boldsymbol{\nabla}^2 p$ as

$$\begin{aligned}\left(\boldsymbol{\nabla}^2 p\right)^{\mathrm{i,j,k}} =& \frac{p^{\mathrm{i\text{-}1,j,k}} + p^{\mathrm{i+1,j,k}}}{\delta x^2} + \frac{p^{\mathrm{i,j\text{-}1,k}} + p^{\mathrm{i,j+1,k}}}{\delta y^2} + \frac{p^{\mathrm{i,j,k\text{-}1}} + p^{\mathrm{i,j,k+1}}}{\delta z^2} \\ & - 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) p^{\mathrm{i,j,k}} \quad ,\end{aligned} \tag{3.23}$$

and $\boldsymbol{\nabla}\cdot\boldsymbol{u}$ as

$$\left(\boldsymbol{\nabla}\cdot\boldsymbol{u}\right)^{\mathrm{i,j,k}} = \frac{u_x^{\mathrm{i+\frac{1}{2},j,k}} - u_x^{\mathrm{i\text{-}\frac{1}{2},j,k}}}{\delta x} + \frac{u_y^{\mathrm{i,j+\frac{1}{2},k}} - u_y^{\mathrm{i,j\text{-}\frac{1}{2},k}}}{\delta y} + \frac{u_z^{\mathrm{i,j,k+\frac{1}{2}}} - u_z^{\mathrm{i,j,k\text{-}\frac{1}{2}}}}{\delta z} \quad . \tag{3.24}$$

Using the spatial discretizations in Equations (3.23) and (3.24), we can formulate Equation (3.22) as a linear system

$$\boldsymbol{K_p} p^n = \boldsymbol{b_p} \quad . \tag{3.25}$$

We again perform the multiplication of $\boldsymbol{K_p} p$ on the grid and use the conjugate gradient algorithm to solve Equation (3.25) (see Algorithm 7).

Finally we time integrate Equation (2.26) and project $\boldsymbol{\nabla} p^n$ on $\boldsymbol{u}^{(3)}$ to obtain divergence-free $\boldsymbol{u}^{n+1}$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{(3)} - \frac{\delta t}{\rho}\boldsymbol{\nabla} p^n \quad . \tag{3.26}$$

The details of the pressure projection step are provided in Algorithm 8.

3.4 Surface Tracking

The free surface of the fluid is tracked by using the particle level-set method described in Enright et al. [2002], but with the substantially faster, though less accurate, method detailed in Enright et al. [2004]. In level-set methods the surface S is defined by an implicit function

$\Phi(\boldsymbol{x})$ whose zero level-set, i.e. $\Phi(\boldsymbol{x}) = 0$, represents the surface. Any function would do, but using a signed distance function where

$$\Phi(\boldsymbol{x}) = \begin{cases} \min\limits_{\boldsymbol{p} \in S} \|\boldsymbol{x} - \boldsymbol{p}\|, & \text{if } \boldsymbol{x} \text{ is outside the fluid} \\ 0, & \text{if } \boldsymbol{x} \text{ is on the surface } S \\ -\min\limits_{\boldsymbol{p} \in S} \|\boldsymbol{x} - \boldsymbol{p}\|, & \text{if } \boldsymbol{x} \text{ is inside the fluid} \end{cases} \tag{3.27}$$

has certain advantages. First, a signed distance function is smooth everywhere, except the medial axes, and can be differentiated. Second, $\boldsymbol{\nabla}\Phi(\boldsymbol{x})$ is the unit normal to the surface at any point $\boldsymbol{x}$ on the surface. Additionally, for any point $\boldsymbol{x}$ in space, away from the surface, $\boldsymbol{\nabla}\Phi(\boldsymbol{x})$ is the direction to the closest point $\boldsymbol{p}$ on the surface to the point $\boldsymbol{x}$. In fact, the closest surface point $\boldsymbol{p}$ for any given point $\boldsymbol{x}$ in space can be easily computed by $\boldsymbol{p} = \boldsymbol{x} - \Phi(\boldsymbol{x})\|\boldsymbol{\nabla}\Phi(\boldsymbol{x})\|$.

To represent the level-set in our simulation we discretize the distance values Φ on a separate grid. Enright et al. [2004] note that the method is susceptible to volume loss, and we found this behavior to be problematic for some of our examples that involve fixed, small amounts of fluid. We were able to ameliorate this problem somewhat by using a level-set grid with twice the fluid grid's resolution, and that is staggered with respect to the fluid grid as shown in Figure 3.9. This scheme places level-set grid centers on the cell centers, face centers, edge centers, and nodes of the fluid grid. In addition to helping to prevent volume loss by locating level-set values where velocity boundary constraints are enforced, the higher resolution also benefits the rendered surfaces.

Once the signed distance function $\Phi(\boldsymbol{x})$ is initialized on the grid given the initial state, the surface can be tracked by advecting the distance values on the grid by solving

$$\frac{\partial \Phi}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla}\Phi = 0 \tag{3.28}$$

We again use the semi-Lagrangian advection algorithm to advect Φ values on the grid. However, simply advecting the Φ values can cause dissipation in time. Moreover, since $\Phi(\boldsymbol{x})$ does not have a reliable derivative near medial axes, relying on the grid Φ alone can cause problems in areas with high curvature. To fix this, Enright et al. [2002; 2004] maintain a dense set of particles in a narrow band of three cells on both sides of the surface, and use these particles to correct the surface in these problem areas. To do this, after Φ values get advected, the particles are moved forward along the velocity field using the path trace algorithm in Algorithm 2. In smooth areas the particles on both sides of the surface tend to stay on their side and agree where the surface is. However, in problem areas positive particles escape to the negative side and negative particles escape to the positive side of the fluid. The distance values of these escaped particles are then used to correct the errors in the grid Φ values. Since after the advection and the error-correction the resulting Φ field is not guaranteed to be a signed distance field, the fast marching extrapolation

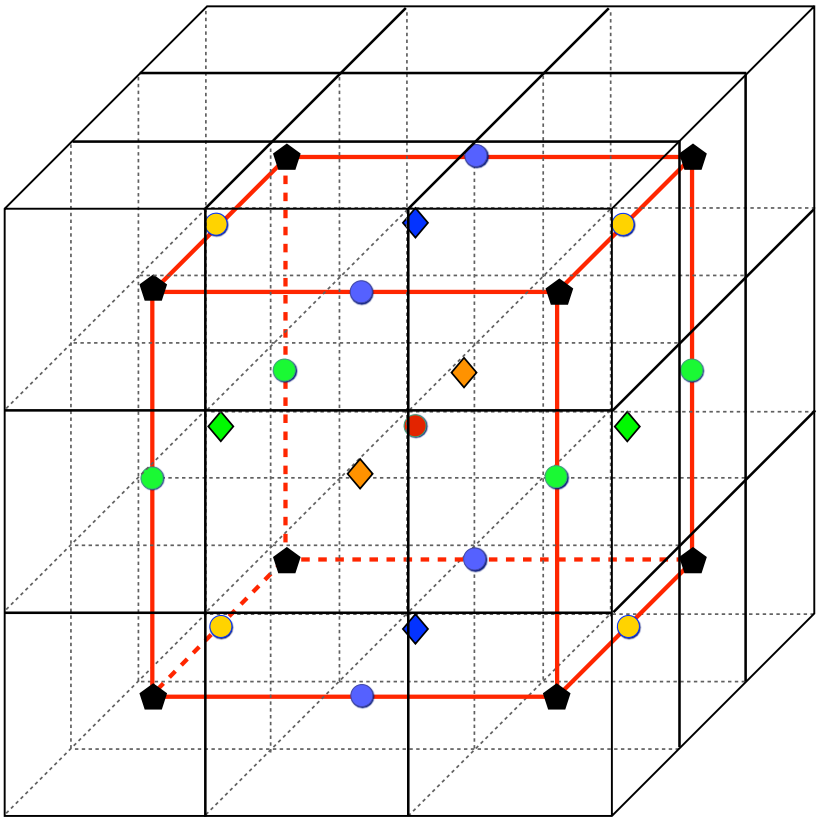

Figure 3.9: The grid for storing the signed distance Φ values (in black). This signed distance grid has higher resolution and is staggered with respect to the computational grid (in red), so that every cell center, face center, edge center and corner on the computational domain is a cell center in the Φ grid and thus has a signed distance value associated with it.

method discussed in Section 3.6 is used to reinitialize signed distance Φ values away from the surface. After the reinitialization step, the location of the zero level-set may change. Therefore the error-correction step is repeated again. Finally the particles are reseeded and reinitialized using the new surface location, if necessary. The high level algorithm of the particle leve-set method for tracking the surface is provided in Algorithm 9. For complete set of details on each step of this method we refer the reader to Enright et al. [2002; 2004] and Osher et al. [2003].

3.5 Updating Strain

At the end of each time step, after the divergence-free velocity field $\boldsymbol{u}^{n+1}$ is computed by the fluid simulator, the strain needs to be updated by solving Equation (2.23). Using the forward Euler method, Equation (2.23) can be integrated as

$$\begin{aligned}
\frac{\boldsymbol{\epsilon}^{n+1}-\boldsymbol{\epsilon}^{n}}{\delta t} = & -(\boldsymbol{u}^{n+1}\cdot\nabla)\boldsymbol{\epsilon}^{n} && (3.29a)\\
& -\boldsymbol{\epsilon}^{n}\cdot\boldsymbol{W}^{n+1}+\boldsymbol{W}^{n+1}\cdot\boldsymbol{\epsilon}^{n} && (3.29b)\\
& +\left(\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1}+\left(\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1}\right)^{\mathsf{T}}\right)/2 && (3.29c)\\
& -\alpha\frac{\boldsymbol{\epsilon}'^{n}}{\|\boldsymbol{\epsilon}'^{n}\|}\max\left(0,\|\boldsymbol{\epsilon}'^{n}\|-\gamma\right) && (3.29d)
\end{aligned}$$

First, like velocity or any other fluid property, the elastic strain must be moved with the fluid in Equation (3.29a). Therefore, each component of the strain tensor is advected using the semi-Lagrangian advection scheme described in Section 3.3.1. In order to avoid the artifacts discussed in Section 2.4, $\boldsymbol{\epsilon}^{n}$ must be rotated by the spin tensor $\boldsymbol{W}^{n+1}=\left(\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1}-(\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1})^{\mathsf{T}}\right)/2$ in Equation (3.29b). Using the discretization of the velocity gradient in Equation (3.3) the skew-symmetric spin tensor $\boldsymbol{W}^{n+1}$ can be computed as

$$\begin{aligned}
W_{xx}^{\mathrm{i,j,k}} &= 0\\
W_{xy}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}}-u_x^{\mathrm{i-\frac{1}{2},j-1,k}}}{2\delta y}-\frac{u_y^{\mathrm{i,j-\frac{1}{2},k}}-u_y^{\mathrm{i-1,j-\frac{1}{2},k}}}{2\delta x}\\
W_{xz}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}}-u_x^{\mathrm{i-\frac{1}{2},j,k-1}}}{2\delta z}-\frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}}-u_z^{\mathrm{i-1,j,k-\frac{1}{2}}}}{2\delta x}\\
W_{yx}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}} &= -W_{xy}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}}\\
W_{yy}^{\mathrm{i,j,k}} &= 0\\
W_{yz}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}} &= \frac{u_y^{\mathrm{i,j-\frac{1}{2},k}}-u_y^{\mathrm{i,j-\frac{1}{2},k-1}}}{2\delta z}-\frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}}-u_z^{\mathrm{i,j-1,k-\frac{1}{2}}}}{2\delta y}\\
W_{zx}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}} &= -W_{xz}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}}\\
W_{zy}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}} &= -W_{yz}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}}\\
W_{zz}^{\mathrm{i,j,k}} &= 0
\end{aligned} \qquad (3.30)$$

Computing the rotation term in Equation (3.29b) requires multiplication of two rank-two tensors $\boldsymbol{\epsilon}^{n}$ and $\boldsymbol{W}^{n+1}$ which can be somewhat inefficient for generalized rank-two tensors on a staggered grid (see [Gerritsma, 1996] for a 2-D case). However since $\boldsymbol{\epsilon}^{n}$ and $\boldsymbol{W}^{n+1}$ are

symmetric and skew-symmetric respectively, computation of $\boldsymbol{A}^n = \boldsymbol{W}^{n+1} \cdot \boldsymbol{\epsilon}^n - \boldsymbol{\epsilon}^n \cdot \boldsymbol{W}^{n+1}$ can be simplified as

$$
\begin{aligned}
A_{xx}^{\mathrm{i,j,k}} &= 2(\epsilon_{xy}^{\mathrm{i,j,k}} W_{xy}^{\mathrm{i,j,k}} + \epsilon_{xz}^{\mathrm{i,j,k}} W_{xz}^{\mathrm{i,j,k}}) \\
A_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} &= -\epsilon_{xx}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} + \epsilon_{xz}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} W_{yz}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \\
&\quad + W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \epsilon_{yy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} + W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \epsilon_{yz}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \\
A_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} &= -\epsilon_{xx}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} - \epsilon_{xy}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} W_{yz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \\
&\quad + W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \epsilon_{yz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} + W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \epsilon_{zz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \\
A_{yx}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} &= A_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \\
A_{yy}^{\mathrm{i,j,k}} &= -2(\epsilon_{xy}^{\mathrm{i,j,k}} W_{xy}^{\mathrm{i,j,k}} - \epsilon_{yz}^{\mathrm{i,j,k}} W_{yz}^{\mathrm{i,j,k}}) \\
A_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} &= -\epsilon_{xy}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} W_{xz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} - \epsilon_{yy}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} W_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \\
&\quad - W_{xy}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \epsilon_{xz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} + W_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \epsilon_{zz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \\
A_{zx}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} &= A_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \\
A_{zy}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} &= A_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \\
A_{zz}^{\mathrm{i,j,k}} &= -2(\epsilon_{xz}^{\mathrm{i,j,k}} W_{xz}^{\mathrm{i,j,k}} + \epsilon_{yz}^{\mathrm{i,j,k}} W_{yz}^{\mathrm{i,j,k}})
\end{aligned}
\tag{3.31}
$$

Note that not all the elements of $\boldsymbol{\epsilon}^n$ and $\boldsymbol{W}^{n+1}$ are defined where they are needed to compute the elements of $\boldsymbol{A}^n$. For example $A_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}$ depends on $\epsilon_{xx}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}$ which is stored on a cell center instead of an xy-edge center where $A_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}$ is computed. We can compute $\epsilon_{xx}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}$ by averaging the surrounding cell centered ϵ_{xx} values as $\epsilon_{xx}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} = (\epsilon_{xx}^{\mathrm{i,j,k}} + \epsilon_{xx}^{\mathrm{i\text{-}1,j,k}} + \epsilon_{xx}^{\mathrm{i\text{-}1,j\text{-}1,k}} + \epsilon_{xx}^{\mathrm{i,j\text{-}1,k}})/4$. Similarly $A_{xx}^{\mathrm{i,j,k}}$ depends on $\epsilon_{xy}^{\mathrm{i,j,k}}$ which can be computed by averaging the surrounding edge centered ϵ_{xy} values as $\epsilon_{xy}^{\mathrm{i,j,k}} = (\epsilon_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} + \epsilon_{xy}^{\mathrm{i+\frac{1}{2},j\text{-}\frac{1}{2},k}} + \epsilon_{xy}^{\mathrm{i\text{-}\frac{1}{2},j+\frac{1}{2},k}} + \epsilon_{xy}^{\mathrm{i+\frac{1}{2},j+\frac{1}{2},k}})/4$. Another, simpler alternative is to reconstruct all components of both $\boldsymbol{\epsilon}^n$ and $\boldsymbol{W}^{n+1}$ at the cell centers, computing $\boldsymbol{A}^n$ there and redistributing the off-diagonal components back to the edge centers. However this scheme uses an additional ring of tensor values and causes additional smoothing of $\boldsymbol{A}^n$.

Next the strain rate $\boldsymbol{D}^{n+1} = \left(\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1} + (\boldsymbol{\nabla}\,\boldsymbol{u}^{n+1})^{\mathsf{T}}\right)/2$ is integrated into the strain

in Equation (3.29c) which can be computed as

$$
\begin{aligned}
D_{xx}^{\mathrm{i,j,k}} &= \frac{u_x^{\mathrm{i+\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j,k}}}{\delta x} \\
D_{xy}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j-1,k}}}{2\delta y} + \frac{u_y^{\mathrm{i,j-\frac{1}{2},k}} - u_y^{\mathrm{i-1,j-\frac{1}{2},k}}}{2\delta x} \\
D_{xz}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}} &= \frac{u_x^{\mathrm{i-\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j,k-1}}}{2\delta z} + \frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}} - u_z^{\mathrm{i-1,j,k-\frac{1}{2}}}}{2\delta x} \\
D_{yx}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}} &= D_{xy}^{\mathrm{i-\frac{1}{2},j-\frac{1}{2},k}} \\
D_{yy}^{\mathrm{i,j,k}} &= \frac{u_y^{\mathrm{i,j+\frac{1}{2},k}} - u_y^{\mathrm{i,j-\frac{1}{2},k}}}{\delta y} \\
D_{yz}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}} &= \frac{u_y^{\mathrm{i,j-\frac{1}{2},k}} - u_y^{\mathrm{i,j-\frac{1}{2},k-1}}}{2\delta z} + \frac{u_z^{\mathrm{i,j,k-\frac{1}{2}}} - u_z^{\mathrm{i,j-1,k-\frac{1}{2}}}}{2\delta y} \\
D_{zx}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}} &= D_{xz}^{\mathrm{i-\frac{1}{2},j,k-\frac{1}{2}}} \\
D_{zy}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}} &= D_{yz}^{\mathrm{i,j-\frac{1}{2},k-\frac{1}{2}}} \\
D_{zz}^{\mathrm{i,j,k}} &= \frac{u_z^{\mathrm{i,j,k+\frac{1}{2}}} - u_z^{\mathrm{i,j,k-\frac{1}{2}}}}{\delta z}
\end{aligned}
\tag{3.32}
$$

Finally the strain is decayed in Equation (3.29d) based on decay rate α and plastic limit γ. Note that as we did in computing Equation (3.29b), Equation (3.29d) needs to be computed for each component separately. Therefore, in order to compute the Frobenius norm $\|\boldsymbol{\epsilon}'^n\| = [(\epsilon'^n_{xx})^2 + (\epsilon'^n_{yy})^2 + (\epsilon'^n_{zz})^2 + 2(\epsilon'^n_{xy})^2 + 2(\epsilon'^n_{xz})^2 + 2(\epsilon'^n_{yz})^2]^{\frac{1}{2}}$, which needs all of the components of $\boldsymbol{\epsilon}'$ to be present, missing components need to be interpolated as needed by averaging the surrounding values. The details of updating the strain solving Equation (3.29) are provided in Algorithms 10-14.

3.6 Setting Boundary Conditions

To impose the boundary conditions, at the beginning of every sub-step of the simulation explained above, we go through the set of non-free cell centers, faces and edges, and set their values to boundary values specified in Section 2.6. To resolve the solid boundaries, we voxelize the colliders at each time-step on the computational grid by labeling every cell that is inside the collider as a SOLID cell. We also label all the faces between the fluid and the solids as a WALL face. The velocity component at the WALL faces is set to the velocity of the collider to satisfy Equation (2.31). The tangential components of the velocity parallel to the WALL face inside the SOLID cell are set to the corresponding tangential components of the velocity of the fluid across the WALL face as shown in Figure 3.10. If no-slip boundary

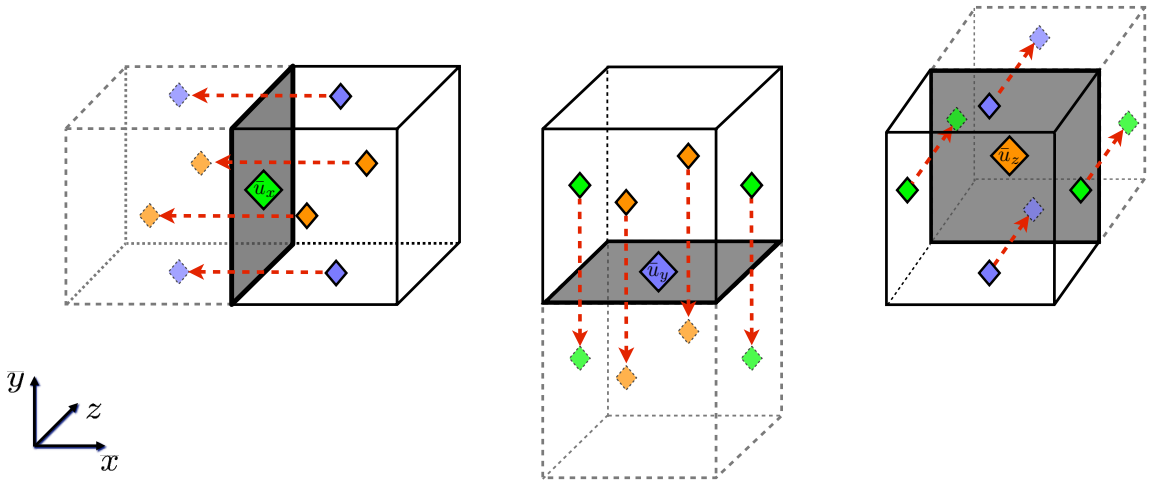

Figure 3.10: Imposing velocity boundary conditions on face aligned solid walls (shaded gray). Velocity component on the WALL face is set to solid collider velocity $\bar{\boldsymbol{u}}$. Velocity components tangent to solid face are set to the corresponding velocity component on the other side of the face (+ for slip, − for no-slip).

condition is chosen, the tangential boundary velocities are multiplied by -1 so that the net tangential velocity at the WALL face is zero[1]. Setting the velocity boundary conditions at the free surface is more involved, because the surface is not necessarily aligned with the grid faces. To satisfy the boundary condition in Equation (2.35) we need to ensure that the velocity components stay constant across the fluid's surface. We achieve this by extrapolating the velocity away from the surface by setting each component to the closest one on the surface of the fluid. We use the efficient fast marching method described in [Enright *et al.*, 2002; 2004] to extrapolate the velocity components into a narrow band of cells across the surface. The details of setting the velocity boundary conditions and the velocity extrapolation are provided in Algorithm 15 and Algorithm 16, respectively.

The solid boundary condition for pressure p in Equation (2.33) is satisfied by setting the pressure values at SOLID cell centers to the ones of the fluid across the WALL faces as shown in Figure 3.11. We set the pressure value to zero for all of the remaining non-free cell centers as specified in Equation (2.37) (see Algorithm 17).

To satisfy the solid boundary condition for elastic strain $\boldsymbol{\epsilon}$ specified in Equation (2.34), we first set the cell centered values ϵ_{xx}, ϵ_{yy} and ϵ_{zz} inside the SOLID cell to those on the fluid cell across the WALL face. Then we set the off-diagonal components ϵ_{xy}, ϵ_{xz} and ϵ_{yz} at the edges

[1]For simplicity, in our current implementation, we only record the normal component of the collider velocities and set the tangential components to zero

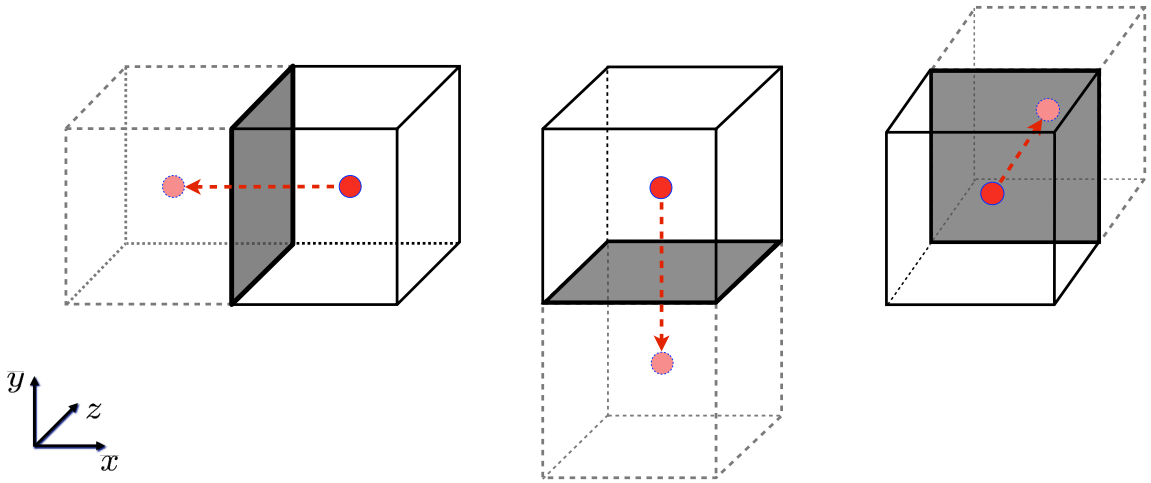

Figure 3.11: Imposing pressure boundary conditions on face aligned solid walls (shaded gray). Cell centered pressure value inside the solid cell is set to the one in the fluid cell across the WALL face.

of the WALL face to zero as shown in Figure 3.12. The free surface boundary condition for $\boldsymbol{\epsilon}$ in Equation (2.36) is satisfied by extrapolating all the components of $\boldsymbol{\epsilon}$ across the surface using the fast marching method. The details of setting the strain boundary conditions and the strain extrapolation are provided in Algorithm 18 and Algorithm 19 respectively.

Note that even though the semi-Lagrangian advection method described in Section 3.3.1 is unconditionally stable, extrapolating the velocity and strain components in a narrow band still poses a constraint on the the time-step. To avoid tracing out of the extrapolated region and risking grabbing an incorrect value for the velocity and the strain, the time-step δt needs to be limited to

$$\delta t < \lambda \frac{\max \|\delta x, \delta y, \delta z\|}{\boldsymbol{u}_{max}} \quad , \tag{3.33}$$

where λ is the width of the extrapolation region ($\lambda = 5$ in our implementation).

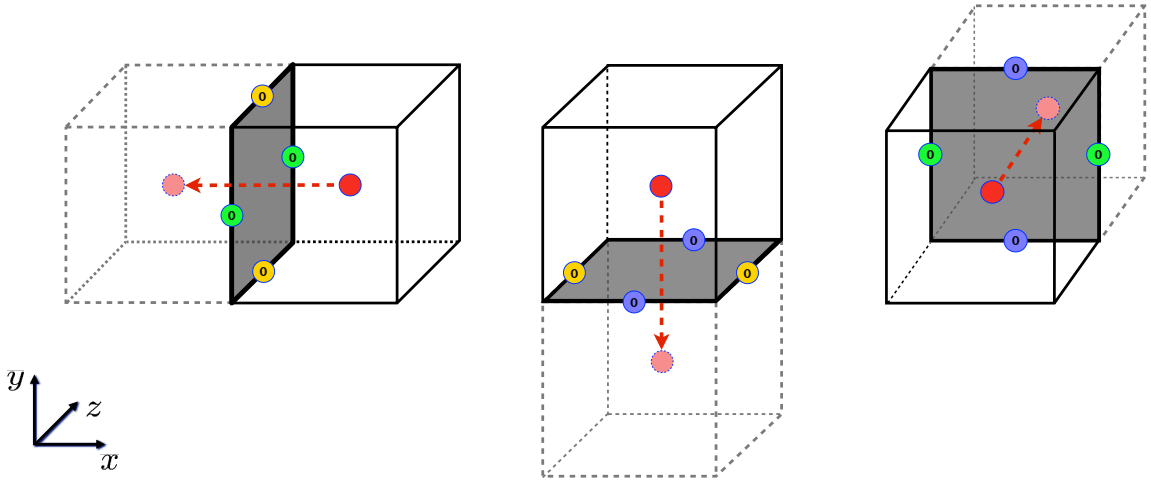

Figure 3.12: Imposing viscoelastic boundary conditions on face aligned solid walls (shaded gray). Cell centered values ϵ_{xx}, ϵ_{yy} and ϵ_{zz} inside the solid cell are set to those in the fluid cell and edge centered components that lie on the wall are set to zero.

Chapter 4

Results and Discussion

We have implemented the method we described in this thesis for modeling viscoelastic behavior and used it to generate several example animations. Most of these examples were selected to illustrate some interesting aspect of viscoelastic fluid behavior. All of the examples shown in this thesis also appear on an accompanying video located at http://graphics.eecs.berkeley.edu/papers/Goktekin-AMF-2004-08, which also contains additional examples.

Figures 4.1 and 4.2 show several splashes that are generated when a fluid sphere is hurled into a tank containing the same material. The motion of the pure fluid example differs substantially from the viscoelastic examples. Additionally, the surfaces of the viscoelastic examples retain evidence of the impact even after motion has stopped. Figure 4.3 shows jets of different fluids sprayed into a closed container. Again, the behavior of simple and viscoelastic fluids differ substantially.

Figures 4.4 and 4.5 illustrate some of the variation in viscoelastic fluid behaviors. In Figure 4.4, fluid draining from a tank forms a thin stream that forms a spiral pattern as it piles. The examples in Figure 4.5 billow up around the downward stream, create a folding pattern, and break apart into gobs. A simply viscous fluid would merely flow out to fill the container.

The drip examples in Figures 4.7 and 4.8 show the behavior of a gob of material that has been stuck to the underside of a horizontal surface. The effect of shape memory created by the elastic forces can be seen in these images. Some of the images in Figure 2.5 show similar behavior that occurs when cubes of different materials are dropped onto a hard surface. The examples with high yield strain, i.e. large γ, behave like deformable solids and bounce.

As shown in Figure 4.9, the method can also model highly deformable, sticky objects that interact with each other. When the spheres collide, their level-set surfaces merge so that they adhere. The fluid retains its momentum, generating the resulting spinning and stretching motion. Close examination shows that the spheres slightly anticipate their collision. This error occurs because the surfaces begin to interact through shared ghost cells.

The images in Figure 4.6 are still images from an animation we produced using this sim-

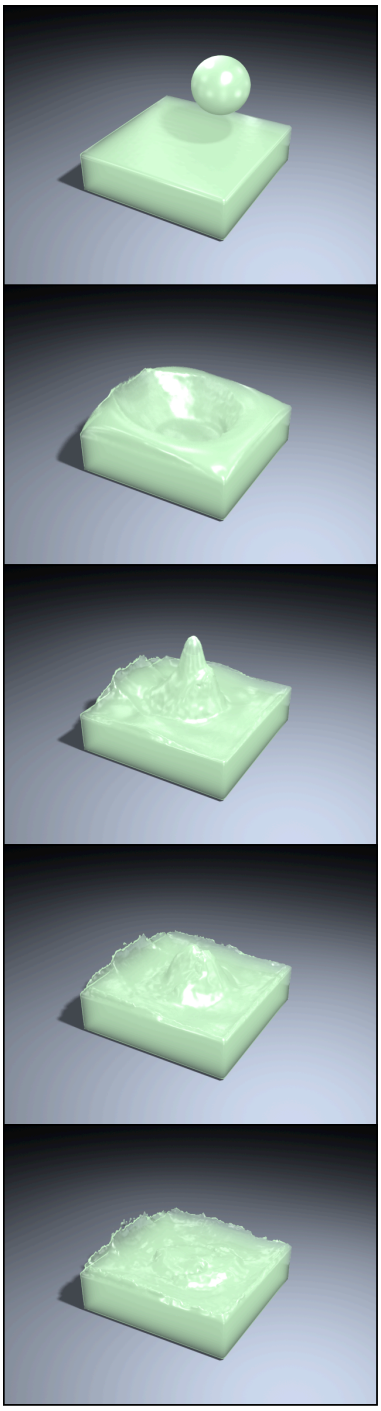

Figure 4.1: A sequence of images showing a splash in a viscoelastic fluid.

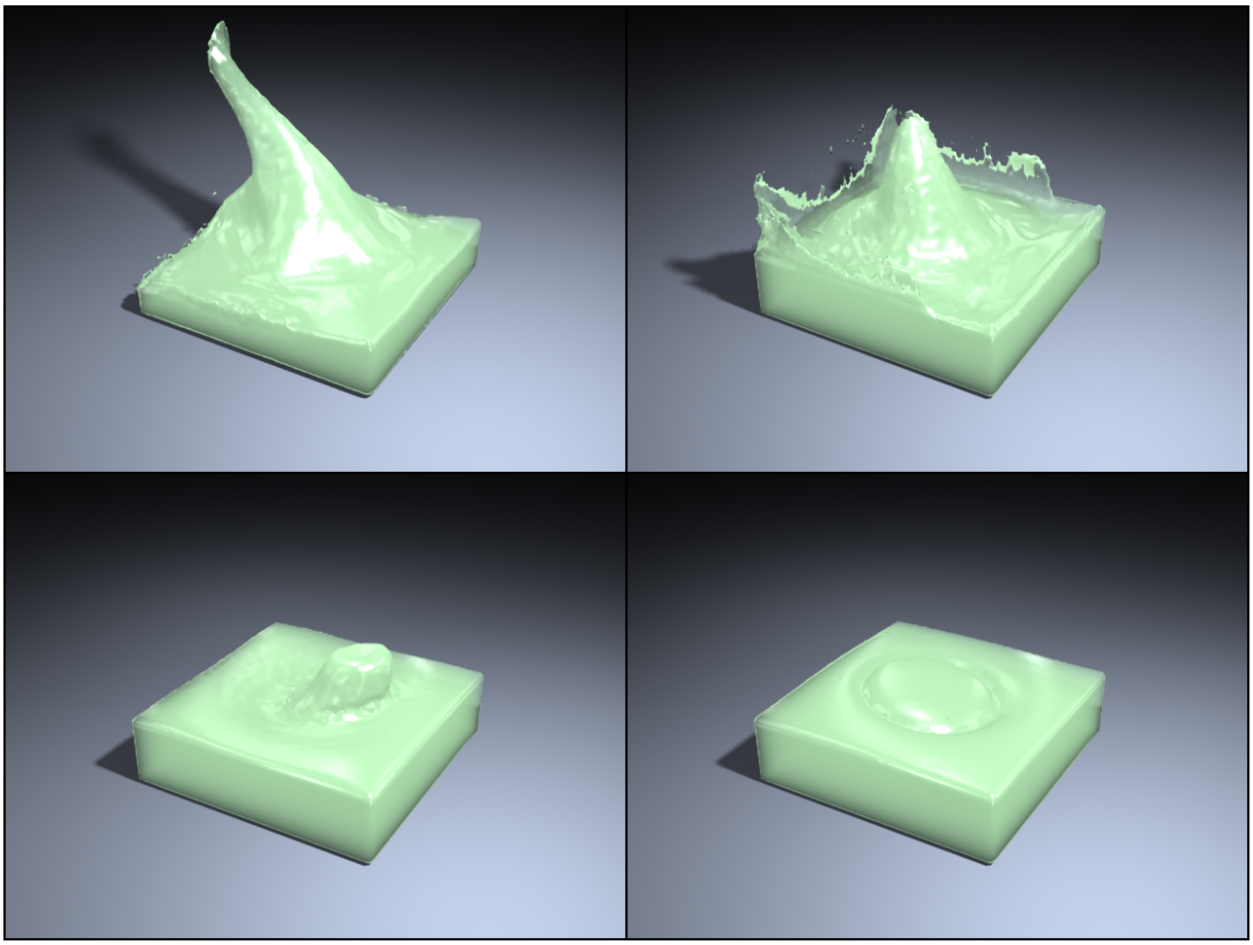

Figure 4.2: These images, along with the ones shown in Figure 4.1, show how splash behavior is affected by elasticity. The upper-left image shows a fluid example. As expected, the Worthington column moves in the direction of the impact. In the viscoelastic examples, the column rises vertically (upper-right) or actually rises back toward the impact direction (lower-left). The lower-right image shows a fluid with both elasticity and high viscosity.

ulation method. These examples demonstrate viscoelastic fluids interacting with interesting geometry.

All of the images were rendered with a standard Newton-iteration based ray marching algorithm implemented in the open source renderer Pixie developed by Okan Arikan.

Some of our examples suffer from noticeable volume loss. This occurs because, while the particle level-set method does a nice job modeling moderately thick volumes of fluids, very thin surfaces, or strands, still have a tendency to vanish. These effects are particularly noticeable visually when the fluid is moving in orderly fashion, as opposed to splashing about chaotically. It is difficult to say if this behavior is a deficiency in our implementation or a

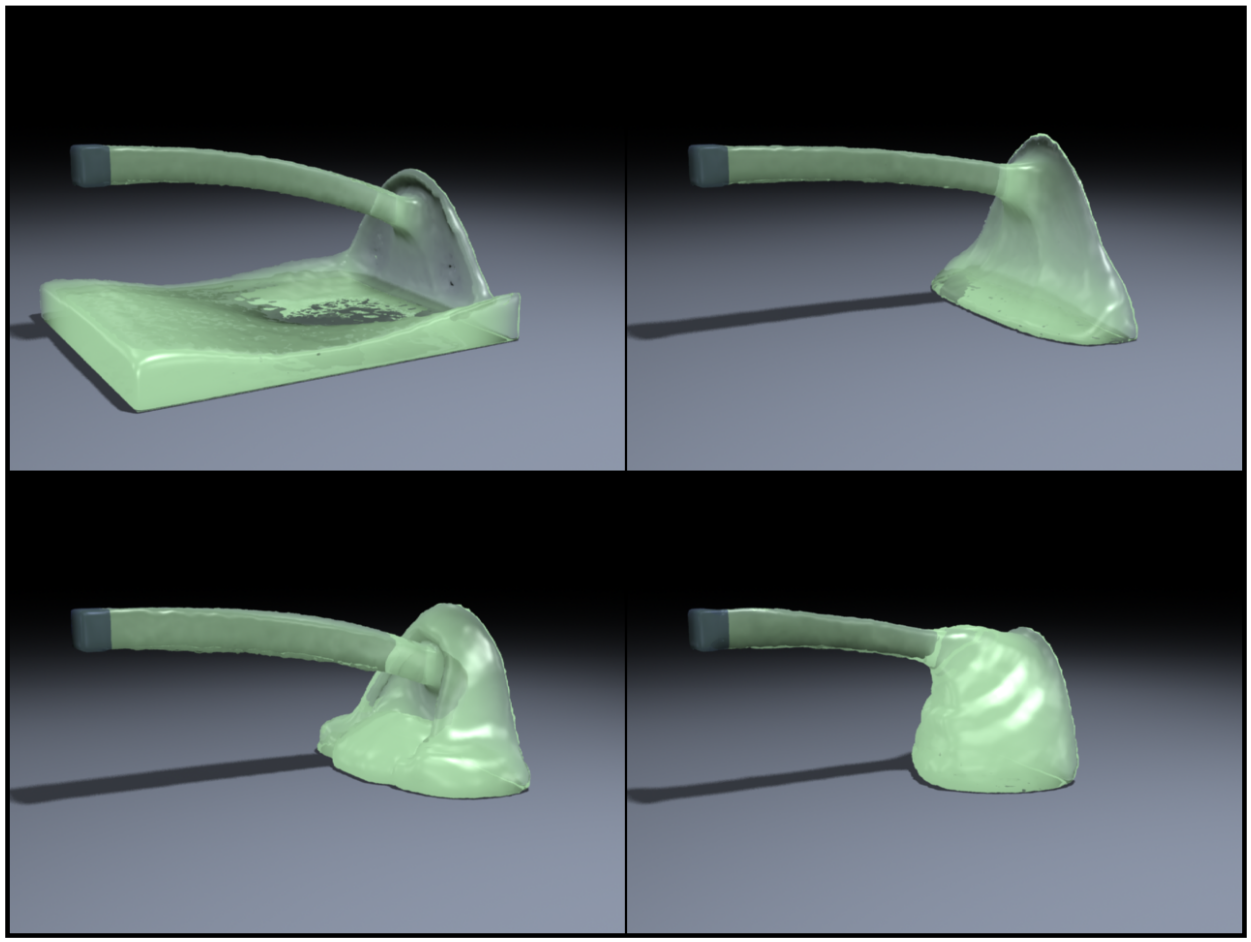

Figure 4.3: Examples of fluid being sprayed into a container. The way different fluids flow or pile in the container varies significantly.

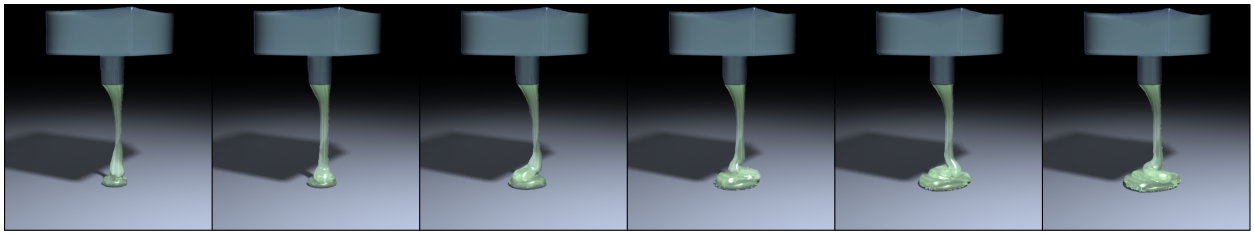

Figure 4.4: A sequence of images showing a viscoelastic fluid draining from a tank. The stream spirals in a fashion characteristic of viscoelastic fluids.

limitation of the surface tracking method.

The speed of this simulation method is approximately the same with and without the

addition of elastic forces. For example, one of the falling cube examples on a 40^3 grid requires about half an hour of computation per second of animation on a single 3 GHz Pentium 4 processor. We are using an explicit integration method for the viscous and elastic forces, so very high viscous or elastic coefficients would probably cause stability problems and force smaller time steps. If it became an issue, that difficulty could be ameliorated somewhat with an implicit integration scheme.

The material can be made to adhere to or slip off of boundary surfaces by adjusting the velocity or pressure constraints enforced along closed boundaries. However, in our current implementation all fluids will stick to each other because different surface components merge when they collide. For the fluids we show in our examples, this behavior is a desirable feature. However, for non-sticky materials, like cold gelatin, it would be undesirable.

To a large extent, our method for incorporating elastoplastic terms does not depend on the underlying fluid simulation method, and one could easily adapt the method to other fluid simulation techniques such as Smoothed-Particle Hydrodynamics (SPH) or Fluid-Implicit-Particle (FLIP). Furthermore, we found that once we already had a working fluid simulation, adding the elastoplastic terms was fairly easy.

In addition to combining fluid and solid properties, there are many ways in which real materials differ from an ideal solid or fluid. For example, solids may exhibit nonlinear stress-to-strain relations, and fluids may exhibit time, pressure, or strain-rate dependent viscosity. These same deviations from the idealized model can occur for viscoelastic fluids. Useful modeling of some materials, such as biological tissues, can require complex constitutive relations. However, many graphics applications have been well served by simple elastic and simple viscous relations, and we have found them useful for viscoelastic fluids as well. For applications where more complex relations are called for, we note that our method for

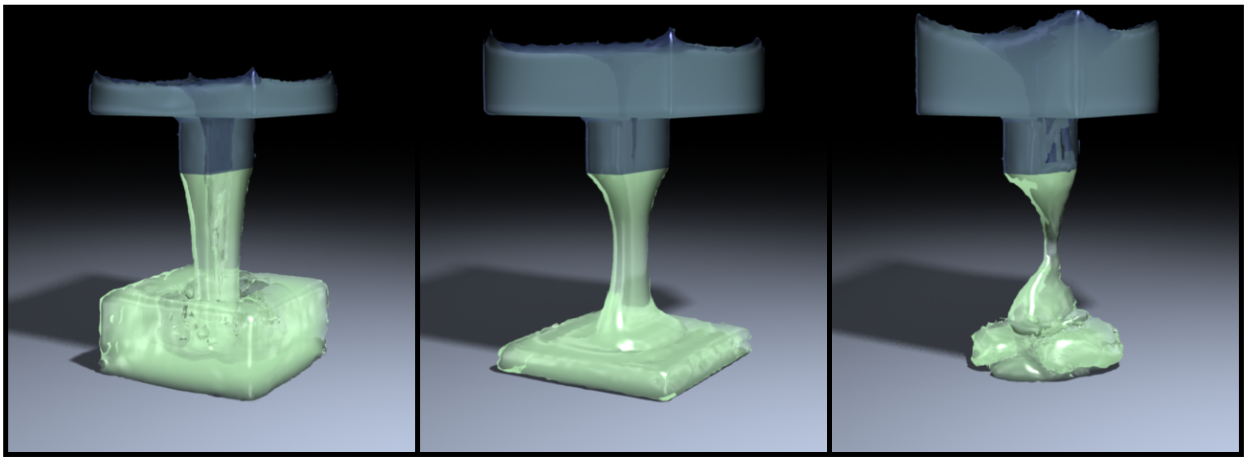

Figure 4.5: These additional examples of viscoelastic fluids draining from a tank show a range of different behaviors.

including elastic forces in a Eulerian fluid simulation are largely independent of the employed constitutive relations.

Finally, we note that while the method we present can model a wide range of phenomena, many real materials can demonstrate behaviors not captured by this model. Biological fluids, such as blood, can exhibit many interesting effects that arise from their microscopic structure. Even relatively simple polymer suspensions can demonstrate behavior that can only be roughly captured with this model.

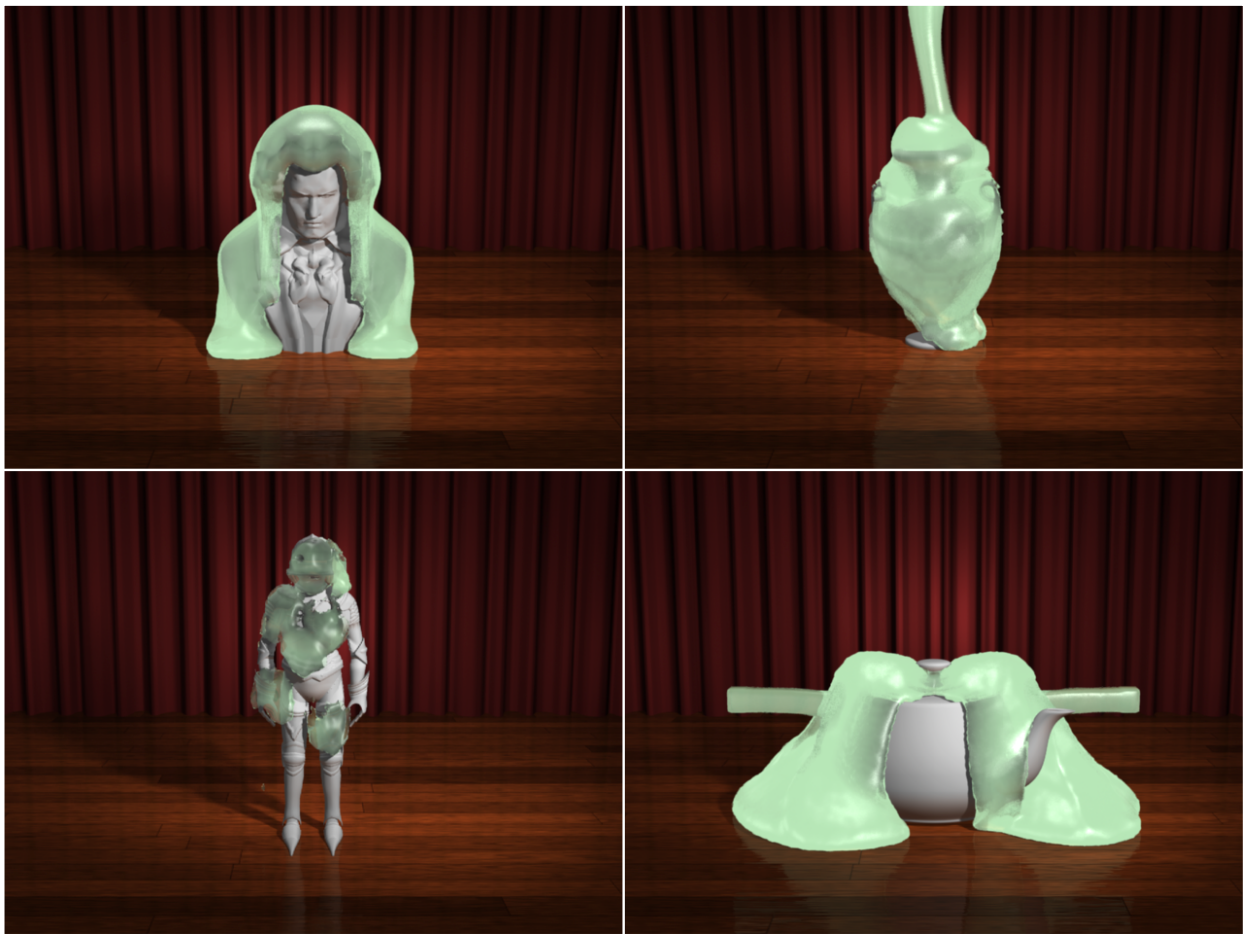

Figure 4.6: Images from the animation *Gratuitous Goop*, which was produced using the methods described in this paper and appeared in the SIGGRAPH 2004 Electronic Theater.

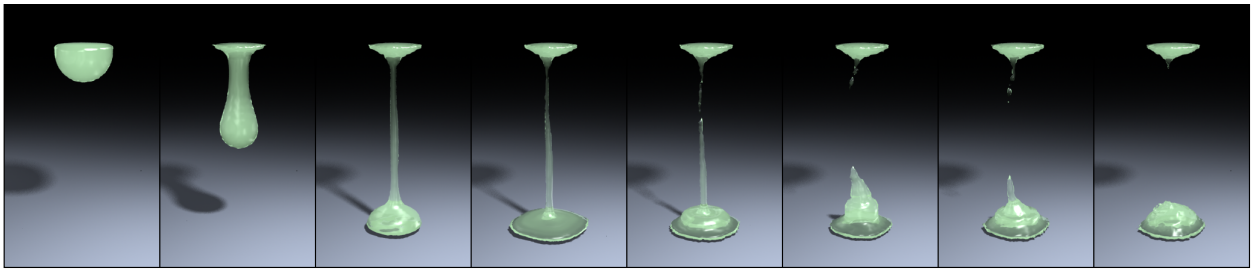

Figure 4.7: A sequence of images showing a viscoelastic fluid dripping off of a surface to which it has adhered.

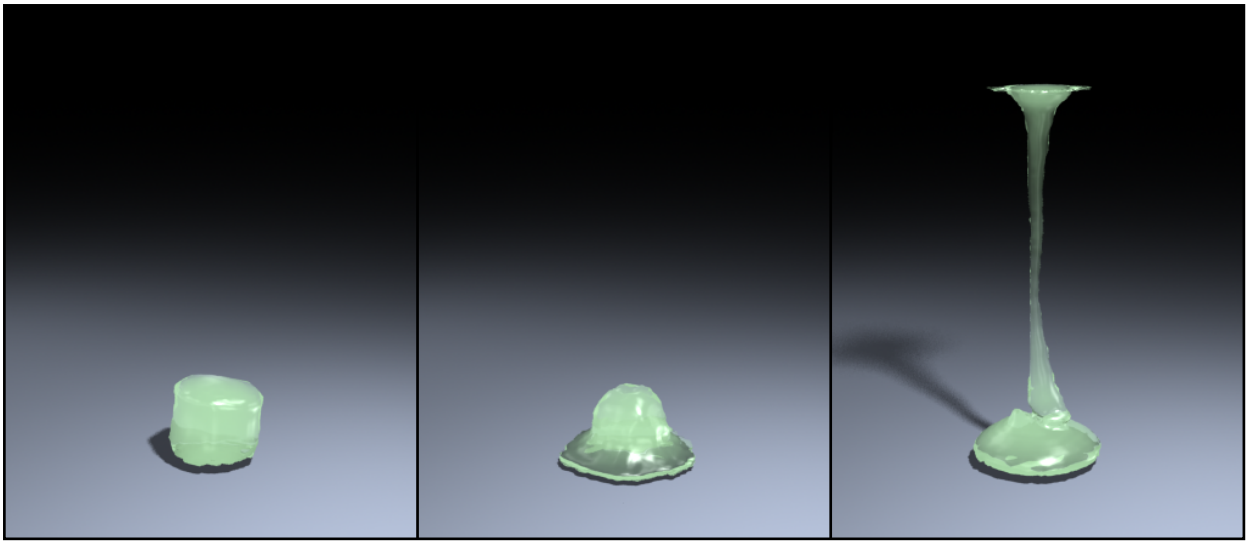

Figure 4.8: Other drip examples. The first is highly elastic and falls off as a single blob. The second does the same as well, but splats because it is less rigid. The final example demonstrates a characteristic swirling behavior.

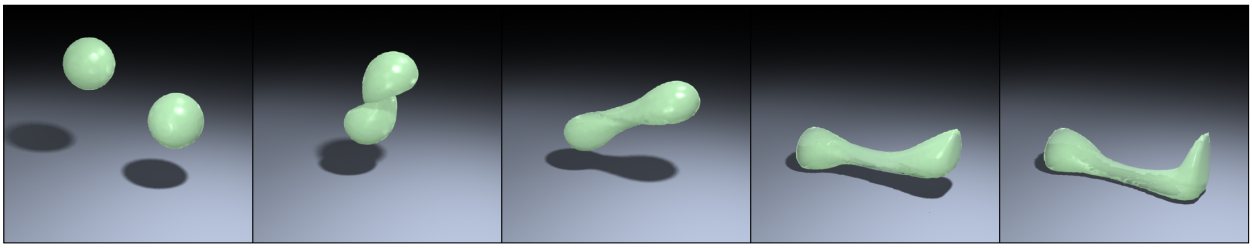

Figure 4.9: Two sticky spheres being thrown into each other so that they adhere together.

Bibliography

[Bargteil *et al.*, 2006] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1), 2006.

[Bingham, 1922] Eugene C. Bingham. *Fluidity And Plasticity*. McGraw-Hill, New York, 1922.

[Bird *et al.*, 1987a] R. Byron Bird, Robert C. Armstrong, and Ole Hassanger. *Dynamics of Polymeric Liquids*, volume 1. John Wiley & Sons, New York, 1987.

[Bird *et al.*, 1987b] R. Byron Bird, Robert C. Armstrong, and Ole Hassanger. *Dynamics of Polymeric Liquids*, volume 2. John Wiley & Sons, New York, 1987.

[Bonito *et al.*, 2006] Andrea Bonito, Marco Picasso, and Manuel Laso. Numerical simulation of 3d viscoelastic flows with free surfaces. *J. Comput. Phys.*, 215(2):691–716, 2006.

[Bridson, 2008] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, Ltd., Wellesley, 2008.

[Cani and Desbrun, 1997] Marie-Paule Cani and Mathieu Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):39–50, January 1997.

[Carlson *et al.*, 2002] Mark Carlson, Peter J. Mucha, R. Brooks Van Horn III, and Greg Turk. Melting and flowing. In *the ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 167–174, July 2002.

[Chorin, 1968] A.J. Chorin. Numerical solution of the Navier-Stokes Equations. *Math. Comp.*, 22:745–762, 1968.

[Desbrun and Cani, 1996] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation 1996*, pages 61–76, August 1996.

[Desbrun and Gascuel, 1995] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. In *the Proceedings of SIGGRAPH 95*, pages 287–290, August 1995.

[Enright *et al.*, 2002] Douglas P. Enright, Stephen R. Marschner, and Ronald P. Fedkiw. Animation and rendering of complex water surfaces. In *the Proceedings of ACM SIGGRAPH 2002*, pages 736–744, July 2002.

[Enright *et al.*, 2004] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures*, 2004.

[Foster and Fedkiw, 2001] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *the Proceedings of ACM SIGGRAPH 2001*, pages 23–30, August 2001.

[Foster and Metaxas, 1996] Nick Foster and Demitri Metaxas. Realistic animation of liquids. In *Graphics Interface 1996*, pages 204–212, May 1996.

[Fung, 1965] Y. C. Fung. *Foundations of Solid Mechanics*. Prentice-Hall, Englewood Cliffs, N.J., 1965.

[Gerritsma, 1996] M. I. Gerritsma. *Time dependent numerical simulations of a viscoelastic fluid on a staggered grid*. PhD thesis, Rijksuniversiteit Groningen, 1996.

[Goktekin *et al.*, 2004] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. A method for animating viscoelastic fluids. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004)*, 23(3):463–468, 2004.

[Goktekin *et al.*, 2007] Tolga G. Goktekin, Jon Reisch, Darwyn Peachey, and Apurva Shah. An effects recipe for rolling a dough, cracking an egg and pouring a sauce. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Han and Reddy, 1999] W. Han and B. D. Reddy. *Plasticity: Mathematical Theory and Numerical Analysis*. Interdisciplinary Applied Mathematics. Springer-Verlag, New York, 1999.

[Harlow and Welch, 1965] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids*, 8:2182–2189, 1965.

[Hohenemser and Prager, 1932] K. Hohenemser and W. Prager. ber die anstze der mechanik isotroper kontinua. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift fr Angewandte Mathematik und Mechanik,*, 12:216–226, 1932.

[Irving, 2007] Geoffrey Irving. *Methods for the physically based simulation of solids and fluids*. PhD thesis, Stanford University, 2007.

[Mao and Yang, 2006] Hai Mao and Yee-Hong Yang. Particle-based non-newtonian fluid animation with heating effects. Technical Report TR06–12, University of Alberta, 2006.

[Maxwell, 1867] J. C. Maxwell. On the dynamical theory of gases. *Philosophical Transactions of the Royal Society*, 157:49–88, 1867.

[Müller *et al.*, 2003] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *the ACM SIGGRAPH 2003 Symposium on Computer Animation*, pages 154–159, August 2003.

[Nixon and Lobb, 2002] Daniel Nixon and Richard Lobb. A fluid-based soft-object model. *IEEE Computer Graphics & Applications*, 22(4):68–75, July 2002.

[O'Brien *et al.*, 2002] J. F. O'Brien, Adam W. Bargteil, and J. K. Hodgins. Graphical modeling and animation of ductile fracture. In *the Proceedings of ACM SIGGRAPH 2002*, pages 291–294, July 2002.

[Oldroyd, 1947] J. Oldroyd. A rational formulation of the equations of plastic flow for a bingham solid. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43:100–105, 1947.

[Osakada, 2010] K. Osakada. History of plasticity and metal forming analysis. *Journal of Materials Processing Technology*, 210(11):1436–1454, August 2010.

[Osher and Fedkiw, 2003] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, 2003.

[Premoze *et al.*, 2003] Simon Premoze, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross Whitaker. Particle–based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, September 2003.

[Ruilova, 2007] Allen Ruilova. Creating realistic cg honey. In *ACM SIGGRAPH 2007 posters*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Shewchuk, 1994] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, 1994.

[Stam, 1999] Jos Stam. Stable fluids. In *the Proceedings of ACM SIGGRAPH 1999*, pages 121–128, August 1999.

[Stora *et al.*, 1999] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface 99*, pages 203–210, June 1999.

[Terzopoulos and Fleischer, 1988a] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–331, 1988.

[Terzopoulos and Fleischer, 1988b] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. *SIGGRAPH Comput. Graph.*, 22:269–278, June 1988.

[Terzopoulos *et al.*, 1989] Demetri Terzopoulos, John Platt, and Kurt Fleischer. From goop to glop: heating and melting deformable models. In *Graphics Interface*, 1989.

[Tomé *et al.*, 2002] M.F. Tomé, N. Mangiavacchi, J.A. Cuminato, A. Castelo, and S. McKee. A finite difference technique for simulating unsteady viscoelastic free surface flows. *Journal of Non-Newtonian Fluid Mechanics*, 106:61–106, 2002.

[von Mises, 1913] Richard von Mises. Mechanik der festen Körper im plastisch- deformablen Zustand. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1913:582–592, 1913.

Appendices

Appendix A

Algorithms

Algorithm 1 Computing $\boldsymbol{f}_e^n = \mu_e \boldsymbol{\nabla} \cdot \boldsymbol{\epsilon}^n$ on the grid

procedure ElasticForces($\boldsymbol{\epsilon}$)
  ApplyStrainBC($\boldsymbol{\epsilon}$)
  for all free x-face $(\text{i-}\frac{1}{2}\text{,j,k})$ **do**
    $(f_e)_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow \mu_e\left(\frac{\epsilon_{xx}^{\text{i,j,k}} - \epsilon_{xx}^{\text{i-1,j,k}}}{\delta x} + \frac{\epsilon_{xy}^{\text{i-}\frac{1}{2}\text{,j+}\frac{1}{2}\text{,k}} - \epsilon_{xy}^{\text{i-}\frac{1}{2}\text{,j-}\frac{1}{2}\text{,k}}}{\delta y} + \frac{\epsilon_{xz}^{\text{i-}\frac{1}{2}\text{,j,k+}\frac{1}{2}} - \epsilon_{xz}^{\text{i-}\frac{1}{2}\text{,j,k-}\frac{1}{2}}}{\delta z}\right)$
  end for
  for all free y-face $(\text{i,j-}\frac{1}{2}\text{,k})$ **do**
    $(f_e)_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow \mu_e\left(\frac{\epsilon_{xy}^{\text{i+}\frac{1}{2}\text{,j-}\frac{1}{2}\text{,k}} - \epsilon_{xy}^{\text{i-}\frac{1}{2}\text{,j-}\frac{1}{2}\text{,k}}}{\delta x} + \frac{\epsilon_{yy}^{\text{i,j,k}} - \epsilon_{yy}^{\text{i,j-1,k}}}{\delta y} + \frac{\epsilon_{yz}^{\text{i,j-}\frac{1}{2}\text{,k+}\frac{1}{2}} - \epsilon_{yz}^{\text{i,j-}\frac{1}{2}\text{,k-}\frac{1}{2}}}{\delta z}\right)$
  end for
  for all free z-face $(\text{i,j,k-}\frac{1}{2})$ **do**
    $(f_e)_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow \mu_e\left(\frac{\epsilon_{xz}^{\text{i+}\frac{1}{2}\text{,j,k-}\frac{1}{2}} - \epsilon_{xz}^{\text{i-}\frac{1}{2}\text{,j,k-}\frac{1}{2}}}{\delta x} + \frac{\epsilon_{yz}^{\text{i,j+}\frac{1}{2}\text{,k-}\frac{1}{2}} - \epsilon_{yz}^{\text{i,j-}\frac{1}{2}\text{,k-}\frac{1}{2}}}{\delta y} + \frac{\epsilon_{zz}^{\text{i,j,k}} - \epsilon_{zz}^{\text{i,j,k-1}}}{\delta z}\right)$
  end for
  return $\boldsymbol{f_e}$
end procedure

Algorithm 2 Advecting the velocity using the Semi-Lagrangian advection method

procedure PATHTRACE($\boldsymbol{x}$, $\boldsymbol{u}$)
  $\boldsymbol{x}_{mid} \leftarrow \boldsymbol{x} + \frac{\delta t}{2}\,\boldsymbol{u}$
  $\boldsymbol{x}_f \leftarrow \boldsymbol{x} - \delta t \boldsymbol{u}(\boldsymbol{x}_{mid})$
  return $\boldsymbol{x}_f$
end procedure

procedure ADVECTVELOCITY($\boldsymbol{u}$)
  APPLYVELOCITYBC($\boldsymbol{u}$)
  $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u}$
  for all free x-face $\left(\text{i-}\frac{1}{2}\text{,j,k}\right)$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\text{i-}\frac{1}{2}\text{,j,k}})$
    $\boldsymbol{x}_f \leftarrow \text{PATHTRACE}(\boldsymbol{x}^{\text{i-}\frac{1}{2}\text{,j,k}}, \boldsymbol{u}_s)$
    $\hat{u}_x \leftarrow u_x(\boldsymbol{x}_f)$
  end for
  for all free y-face $\left(\text{i,j-}\frac{1}{2}\text{,k}\right)$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\text{i,j-}\frac{1}{2}\text{,k}})$
    $\boldsymbol{x}_f \leftarrow \text{PATHTRACE}(\boldsymbol{x}^{\text{i,j-}\frac{1}{2}\text{,k}}, \boldsymbol{u}_s)$
    $\hat{u}_y \leftarrow u_y(\boldsymbol{x}_f)$
  end for
  for all free z-face $\left(\text{i,j,k-}\frac{1}{2}\right)$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\text{i,j,k-}\frac{1}{2}})$
    $\boldsymbol{x}_f \leftarrow \text{PATHTRACE}(\boldsymbol{x}^{\text{i,j,k-}\frac{1}{2}}, \boldsymbol{u}_s)$
    $\hat{u}_z \leftarrow u_z(\boldsymbol{x}_f)$
  end for
  return $\hat{\boldsymbol{u}}$
end procedure

Algorithm 3 Adding gravity and elastic forces

procedure AddForces($\boldsymbol{u}$)
  $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u}$
  for all free x-face $(\text{i-}\frac{1}{2}\text{,j,k})$ **do**
    $\hat{u}_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow u_x^{\text{i-}\frac{1}{2}\text{,j,k}} + \delta t \left(g_x + \frac{1}{\rho} f_{e_x}{}^{\text{i-}\frac{1}{2}\text{,j,k}} \right)$
  end for
  for all free y-face $(\text{i,j-}\frac{1}{2}\text{,k})$ **do**
    $\hat{u}_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow u_y^{\text{i,j-}\frac{1}{2}\text{,k}} + \delta t \left(g_y + \frac{1}{\rho} f_{e_y}{}^{\text{i,j-}\frac{1}{2}\text{,k}} \right)$
  end for
  for all free z-face $(\text{i,j,k-}\frac{1}{2})$ **do**
    $\hat{u}_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow u_z^{\text{i,j,k-}\frac{1}{2}} + \delta t \left(g_z + \frac{1}{\rho} f_{e_z}{}^{\text{i,j,k-}\frac{1}{2}} \right)$
  end for
  return $\hat{\boldsymbol{u}}$
end procedure

Algorithm 4 Conjugate Gradient Algorithm

// Solves $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ given max iteration i_{max} and error tolerance ϵ
procedure ConjugateGradient(MultAx, $\boldsymbol{x}$, $\boldsymbol{b}$, i_{max}, ϵ)
  $i \leftarrow 0$
  $\boldsymbol{r} \leftarrow \boldsymbol{b} - \text{MultAx}(\boldsymbol{A}, \boldsymbol{x})$
  $\boldsymbol{d} \leftarrow \boldsymbol{r}$
  $\delta_{new} \leftarrow \boldsymbol{r}^\mathsf{T}\boldsymbol{r}$
  $\delta_0 \leftarrow \delta_{new}$
  while $i < i_{max}$ **and** $\delta_{new} > \epsilon^2 \delta_0$ **do**
    $\boldsymbol{q} \leftarrow \text{MultAx}(\boldsymbol{A}, \boldsymbol{d})$
    $\alpha = \frac{\delta_{new}}{\boldsymbol{d}^\mathsf{T}\boldsymbol{q}}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\boldsymbol{d}$
    $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha\boldsymbol{q}$
    $\delta_{old} \leftarrow \delta_{new}$
    $\delta_{new} \leftarrow \boldsymbol{r}^\mathsf{T}\boldsymbol{r}$
    $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $\boldsymbol{d} \leftarrow \boldsymbol{r} + \beta\boldsymbol{d}$
    $i \leftarrow i + 1$
  end while
  return $\boldsymbol{x}$
end procedure

Algorithm 5 Computing $\hat{\boldsymbol{u}} = \left(\boldsymbol{I} - \delta t \frac{\mu_v}{\rho} \boldsymbol{\nabla}^2\right) \boldsymbol{u}$ foreach component on the Grid

procedure ComputeKUx($\boldsymbol{u}_x$)
  for all free $u_x^{\text{i-}\frac{1}{2},\text{j},\text{k}}$ **do**
    $\hat{u}_x^{\text{i-}\frac{1}{2},\text{j},\text{k}} \leftarrow -\frac{\mu_v}{\rho}\delta t \left(\frac{u_x^{\text{i-}\frac{3}{2},\text{j},\text{k}} + u_x^{\text{i+}\frac{1}{2},\text{j},\text{k}}}{\delta x^2} + \frac{u_x^{\text{i-}\frac{1}{2},\text{j-1},\text{k}} + u_x^{\text{i-}\frac{1}{2},\text{j+1},\text{k}}}{\delta y^2} + \frac{u_x^{\text{i-}\frac{1}{2},\text{j},\text{k-1}} + u_x^{\text{i-}\frac{1}{2},\text{j},\text{k+1}}}{\delta z^2} \right)$
    $\hat{u}_x^{\text{i-}\frac{1}{2},\text{j},\text{k}} \leftarrow \hat{u}_x^{\text{i-}\frac{1}{2},\text{j},\text{k}} + \left(1 + 2\frac{\mu_v}{\rho}\delta t \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_x^{\text{i-}\frac{1}{2},\text{j},\text{k}})\right)$
  end for
  return $\hat{\boldsymbol{u}}_x$
end procedure
procedure ComputeKUy($\boldsymbol{u}_y$)
  for all free $u_y^{\text{i,j-}\frac{1}{2},\text{k}}$ **do**
    $\hat{u}_y^{\text{i,j-}\frac{1}{2},\text{k}} \leftarrow -\frac{\mu_v}{\rho}\delta t \left(\frac{u_y^{\text{i-1,j-}\frac{1}{2},\text{k}} + u_y^{\text{i+1,j-}\frac{1}{2},\text{k}}}{\delta x^2} + \frac{u_y^{\text{i,j-}\frac{3}{2},\text{k}} + u_y^{\text{i,j+}\frac{1}{2},\text{k}}}{\delta y^2} + \frac{u_y^{\text{i,j-}\frac{1}{2},\text{k-1}} + u_y^{\text{i,j-}\frac{1}{2},\text{k+1}}}{\delta z^2} \right)$
    $\hat{u}_y^{\text{i,j-}\frac{1}{2},\text{k}} \leftarrow \hat{u}_y^{\text{i,j-}\frac{1}{2},\text{k}} + \left(1 + 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_y^{\text{i,j-}\frac{1}{2},\text{k}}\right)$
  end for
  return $\hat{\boldsymbol{u}}_y$
end procedure
procedure ComputeKUz($\boldsymbol{u}_z$)
  for all free $u_z^{\text{i,j,k-}\frac{1}{2}}$ **do**
    $\hat{u}_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow -\frac{\mu_v}{\rho}\delta t \left(\frac{u_z^{\text{i-1,j,k-}\frac{1}{2}} + u_z^{\text{i+1,j,k-}\frac{1}{2}}}{\delta x^2} + \frac{u_z^{\text{i,j-1,k-}\frac{1}{2}} + u_z^{\text{i,j+1,k-}\frac{1}{2}}}{\delta y^2} + \frac{u_z^{\text{i,j,k-}\frac{3}{2}} + u_z^{\text{i,j,k+}\frac{1}{2}}}{\delta z^2} \right)$
    $\hat{u}_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow \hat{u}_z^{\text{i,j,k-}\frac{1}{2}} + \left(1 + 2\frac{\mu_v}{\rho}\delta t \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) u_z^{\text{i,j,k-}\frac{1}{2}}\right)$
  end for
  return $\hat{\boldsymbol{u}}_z$
end procedure

Algorithm 6 Solving the Viscosity

procedure Diffuse($\boldsymbol{u}$)
  ApplyVelocityBC($\boldsymbol{u}$)
  $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u}$
  $\hat{u}_x \leftarrow$ ConguateGradient(ComputeKUx(), $\hat{\boldsymbol{u}}_x, \boldsymbol{u}_x, i_{max}, \epsilon$)
  $\hat{u}_y \leftarrow$ ConguateGradient(ComputeKUy(), $\hat{\boldsymbol{u}}_y, \boldsymbol{u}_y, i_{max}, \epsilon$)
  $\hat{u}_z \leftarrow$ ConguateGradient(ComputeKUz(), $\hat{\boldsymbol{u}}_z, \boldsymbol{u}_z, i_{max}, \epsilon$)
  return $\hat{\boldsymbol{u}}$
end procedure

Algorithm 7 Solving the Pressure Poisson equation

procedure ComputeKp(p)
  for all free $p^{i,j,k}$ **do**
    $\hat{p}^{i,j,k} \leftarrow \frac{p^{i\text{-}1,j,k}+p^{i+1,j,k}}{\delta x^2} + \frac{p^{i,j\text{-}1,k}+p^{i,j+1,k}}{\delta y^2} + \frac{p^{i,j,k\text{-}1}+p^{i,j,k+1}}{\delta z^2} - 2\left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}\right) p^{i,j,k}$
  end for
  return $\hat{p}$
end procedure
procedure ComputeDivU($\boldsymbol{u}$)
  for all free $p^{i,j,k}$ **do**
    $b^{i,j,k} \leftarrow \frac{\rho}{\delta t}\left(\frac{u_x^{i+\frac{1}{2},j,k} - u_x^{i-\frac{1}{2},j,k}}{\delta x} + \frac{u_y^{i,j+\frac{1}{2},k} - u_y^{i,j-\frac{1}{2},k}}{\delta y} + \frac{u_z^{i,j,k+\frac{1}{2}} - u_z^{i,j,k-\frac{1}{2}}}{\delta z}\right)$
  end for
  return b
end procedure
procedure SolvePPE(p, $\boldsymbol{u}$)
  $\hat{p} \leftarrow p$
  $b \leftarrow$ ComputeDivU($\boldsymbol{u}$)
  $\hat{p} \leftarrow$ ConguateGradient(ComputeKp(), $\hat{p}, b, i_{max}, \epsilon$)
  return $\hat{p}$
end procedure

Algorithm 8 Pressure Project $\boldsymbol{u}$ to obtain divergence free $\hat{\boldsymbol{u}}$

procedure PROJECT(p,$\boldsymbol{u}$)
  for all free $u_x^{\text{i-}\frac{1}{2},\text{j},\text{k}}$ **do**
    $\hat{\boldsymbol{u}}^{\text{i-}\frac{1}{2},\text{j},\text{k}} \leftarrow \boldsymbol{u}^{\text{i-}\frac{1}{2},\text{j},\text{k}} - \frac{\delta t}{\rho}\frac{\left(p^{\text{i},\text{j},\text{k}} - p^{\text{i-1},\text{j},\text{k}}\right)}{\delta x}$
  end for
  for all free $u_y^{\text{i},\text{j-}\frac{1}{2},\text{k}}$ **do**
    $\hat{\boldsymbol{u}}^{\text{i},\text{j-}\frac{1}{2},\text{k}} \leftarrow \boldsymbol{u}^{\text{i},\text{j-}\frac{1}{2},\text{k}} - \frac{\delta t}{\rho}\frac{\left(p^{\text{i},\text{j},\text{k}} - p^{\text{i},\text{j-1},\text{k}}\right)}{\delta y}$
  end for
  for all free $u_z^{\text{i},\text{j},\text{k-}\frac{1}{2}}$ **do**
    $\hat{\boldsymbol{u}}^{\text{i},\text{j},\text{k-}\frac{1}{2}} \leftarrow \boldsymbol{u}^{\text{i},\text{j},\text{k-}\frac{1}{2}} - \frac{\delta t}{\rho}\frac{\left(p^{\text{i},\text{j},\text{k}} - p^{\text{i},\text{j},\text{k-1}}\right)}{\delta z}$
  end for
end procedure
procedure PRESSUREPROJECT(p,$\boldsymbol{u}$)
  APPLYVELOCITYBC($\boldsymbol{u}$)
  APPLYPRESSUREBC(p)
  $\hat{p} \leftarrow$ SOLVEPPE($p, \boldsymbol{u}$)
  $\hat{\boldsymbol{u}} \leftarrow$ PROJECT($\hat{p}, \boldsymbol{u}$)
  return $\hat{\boldsymbol{u}}$
end procedure

Algorithm 9 Tracking the surface using the particle level-set method

procedure SURFACETRACK(Φ,$\boldsymbol{u}$)
  $\Phi^{(1)} \leftarrow$ ADVECT($\boldsymbol{u}, \Phi$)
  $P^{(1)} \leftarrow$ PATHTRACEPARTICLES(P)
  $\Phi^{(2)} \leftarrow$ ERRORCORRECT($\Phi^{(1)}, P^{(1)}$)
  $\Phi^{(3)} \leftarrow$ EXTRAPOLATE($\Phi^{(2)}$)
  $\Phi^{(4)} \leftarrow$ ERRORCORRECT($\Phi^{(3)}, P^{(1)}$)
  $P \leftarrow$ RESEEDPARTICLES($\Phi^{(4)}, P^{(1)}$)
  return $\Phi^{(4)}$
end procedure

Algorithm 10 Advecting the strain using the Semi-Lagrangian advection method

procedure AdvectStrain($\boldsymbol{\epsilon}$,$\boldsymbol{u}$)
  ApplyStrainBC($\boldsymbol{\epsilon}$)
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}$
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\mathrm{i,j,k}})$
    $\boldsymbol{x}_f \leftarrow \textsc{PathTrace}(\boldsymbol{x}^{\mathrm{i,j,k}}, \boldsymbol{u}_s)$
    $\hat{\epsilon}_{xx}^{\mathrm{i,j,k}} \leftarrow \hat{\epsilon}_{xx}(\boldsymbol{x}_f)$
    $\hat{\epsilon}_{yy}^{\mathrm{i,j,k}} \leftarrow \hat{\epsilon}_{yy}(\boldsymbol{x}_f)$
    $\hat{\epsilon}_{zz}^{\mathrm{i,j,k}} \leftarrow \hat{\epsilon}_{zz}(\boldsymbol{x}_f)$
  end for
  for all free xy-edge $(\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k})$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}})$
    $\boldsymbol{x}_f \leftarrow \textsc{PathTrace}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}, \boldsymbol{u}_s)$
    $\hat{\epsilon}_{xy}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}} \leftarrow \hat{\epsilon}_{xy}(\boldsymbol{x}_f)$
  end for
  for all free xz-edge $(\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2})$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}})$
    $\boldsymbol{x}_f \leftarrow \textsc{PathTrace}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}, \boldsymbol{u}_s)$
    $\hat{\epsilon}_{xz}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}} \leftarrow \hat{\epsilon}_{xz}(\boldsymbol{x}_f)$
  end for
  for all free yz-edge $(\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2})$ **do**
    $\boldsymbol{u}_s \leftarrow -\boldsymbol{u}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}})$
    $\boldsymbol{x}_f \leftarrow \textsc{PathTrace}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}, \boldsymbol{u}_s)$
    $\hat{\epsilon}_{yz}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}} \leftarrow \hat{\epsilon}_{yz}(\boldsymbol{x}_f)$
  end for
  return $\hat{\boldsymbol{\epsilon}}$
end procedure

Algorithm 11 Rotating the strain

procedure ComputeW($\boldsymbol{u}$)
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $W_{xx}^{\mathrm{i,j,k}}, W_{yy}^{\mathrm{i,j,k}}, W_{zz}^{\mathrm{i,j,k}} \leftarrow 0$
  end for
  for all free xy-edge $(\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k})$ **do**
    $W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \leftarrow \frac{u_x^{\mathrm{i\text{-}\frac{1}{2},j,k}} - u_x^{\mathrm{i\text{-}\frac{1}{2},j\text{-}1,k}}}{2\delta y} - \frac{u_y^{\mathrm{i,j\text{-}\frac{1}{2},k}} - u_y^{\mathrm{i\text{-}1,j\text{-}\frac{1}{2},k}}}{2\delta x}$
  end for
  for all free xz-edge $(\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}})$ **do**
    $W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \leftarrow \frac{u_x^{\mathrm{i\text{-}\frac{1}{2},j,k}} - u_x^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}1}}}{2\delta z} - \frac{u_z^{\mathrm{i,j,k\text{-}\frac{1}{2}}} - u_z^{\mathrm{i\text{-}1,j,k\text{-}\frac{1}{2}}}}{2\delta x}$
  end for
  for all free yz-edge $(\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}})$ **do**
    $W_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \leftarrow \frac{u_y^{\mathrm{i,j\text{-}\frac{1}{2},k}} - u_y^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}1}}}{2\delta z} - \frac{u_z^{\mathrm{i,j,k\text{-}\frac{1}{2}}} - u_z^{\mathrm{i,j\text{-}1,k\text{-}\frac{1}{2}}}}{2\delta y}$
  end for
  return $\boldsymbol{W}$
end procedure
procedure RotateStrain($\boldsymbol{\epsilon}$,$\boldsymbol{u}$)
  ApplyStrainBC($\boldsymbol{\epsilon}$)
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}$
  $\boldsymbol{W} \leftarrow$ ComputeW($\boldsymbol{u}$)
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $\hat{\epsilon}_{xx}^{\mathrm{i,j,k}} \leftarrow 2\delta t(\epsilon_{xy}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{xy}(\boldsymbol{x}^{\mathrm{i,j,k}}) + \epsilon_{xz}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{xz}(\boldsymbol{x}^{\mathrm{i,j,k}}))$
    $\hat{\epsilon}_{yy}^{\mathrm{i,j,k}} \leftarrow -2\delta t(\epsilon_{xy}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{xy}(\boldsymbol{x}^{\mathrm{i,j,k}}) - \epsilon_{yz}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{yz}(\boldsymbol{x}^{\mathrm{i,j,k}}))$
    $\hat{\epsilon}_{zz}^{\mathrm{i,j,k}} \leftarrow -2\delta t(\epsilon_{xz}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{xz}(\boldsymbol{x}^{\mathrm{i,j,k}}) + \epsilon_{yz}(\boldsymbol{x}^{\mathrm{i,j,k}})W_{yz}(\boldsymbol{x}^{\mathrm{i,j,k}}))$
  end for
  for all free xy-edge $(\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k})$ **do**
    $\hat{\epsilon}_{xy} \leftarrow \delta t(-\epsilon_{xx}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}})W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} + \epsilon_{xz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}})W_{yz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}})$
      $+W_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}\epsilon_{yy}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}) + W_{xz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}})\epsilon_{yz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}))$
  end for
  for all free xz-edge $(\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}})$ **do**
    $\hat{\epsilon}_{xz} \leftarrow \delta t(-\epsilon_{xx}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}})W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} - \epsilon_{xy}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}})W_{yz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}})$
      $+W_{xy}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}})\epsilon_{yz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}) + W_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}\epsilon_{zz}(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}))$
  end for
  for all free yz-edge $(\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}})$ **do**
    $\hat{\epsilon}_{yz} \leftarrow \delta t(-\epsilon_{xy}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}})W_{xz}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}) - \epsilon_{yy}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}})W_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}$
      $-W_{xy}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}})\epsilon_{xz}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}) + W_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}\epsilon_{zz}(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}))$
  end for
  return $\hat{\boldsymbol{\epsilon}}$
end procedure

Algorithm 12 Computing strain rate and strain deviation

procedure ComputeD($\boldsymbol{u}$)
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $D_{xx}^{\mathrm{i,j,k}} \leftarrow \frac{u_x^{\mathrm{i+\frac{1}{2},j,k}} - u_x^{\mathrm{i-\frac{1}{2},j,k}}}{\delta x}$
    $D_{yy}^{\mathrm{i,j,k}} \leftarrow \frac{u_y^{\mathrm{i,j+\frac{1}{2},k}} - u_y^{\mathrm{i,j-\frac{1}{2},k}}}{\delta y}$
    $D_{zz}^{\mathrm{i,j,k}} \leftarrow \frac{u_z^{\mathrm{i,j,k+\frac{1}{2}}} - u_z^{\mathrm{i,j,k-\frac{1}{2}}}}{\delta z}$
  end for
  for all free xy-edge $(\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k})$ **do**
    $D_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}} \leftarrow \frac{u_x^{\mathrm{i\text{-}\frac{1}{2},j,k}} - u_x^{\mathrm{i\text{-}\frac{1}{2},j\text{-}1,k}}}{2\delta y} + \frac{u_y^{\mathrm{i,j\text{-}\frac{1}{2},k}} - u_y^{\mathrm{i\text{-}1,j\text{-}\frac{1}{2},k}}}{2\delta x}$
  end for
  for all free xz-edge $(\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}})$ **do**
    $D_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}} \leftarrow \frac{u_x^{\mathrm{i\text{-}\frac{1}{2},j,k}} - u_x^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}1}}}{2\delta z} + \frac{u_z^{\mathrm{i,j,k\text{-}\frac{1}{2}}} - u_z^{\mathrm{i\text{-}1,j,k\text{-}\frac{1}{2}}}}{2\delta x}$
  end for
  for all free yz-edge $(\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}})$ **do**
    $D_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}} \leftarrow \frac{u_y^{\mathrm{i,j\text{-}\frac{1}{2},k}} - u_y^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}1}}}{2\delta z} + \frac{u_z^{\mathrm{i,j,k\text{-}\frac{1}{2}}} - u_z^{\mathrm{i,j\text{-}1,k\text{-}\frac{1}{2}}}}{2\delta y}$
  end for
  return $\boldsymbol{D}$
end procedure
procedure ComputeStrainDeviation($\boldsymbol{\epsilon}$)
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}$
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $dilationE \leftarrow \frac{\epsilon_{xx}^{\mathrm{i,j,k}} + \epsilon_{yy}^{\mathrm{i,j,k}} + \epsilon_{zz}^{\mathrm{i,j,k}}}{3}$
    $\hat{\epsilon}_{xx}^{\mathrm{i,j,k}} \leftarrow \epsilon_{xx} - dilationE$
    $\hat{\epsilon}_{yy}^{\mathrm{i,j,k}} \leftarrow \epsilon_{yy} - dilationE$
    $\hat{\epsilon}_{zz}^{\mathrm{i,j,k}} \leftarrow \epsilon_{yy} - dilationE$
  end for
  return $\hat{\boldsymbol{\epsilon}}$
end procedure

Algorithm 13 Integrating the strain rate and decaying the strain

procedure DECAYSTRAIN($\boldsymbol{\epsilon}$,$\boldsymbol{u}$)
  APPLYSTRAINBC($\boldsymbol{\epsilon}$)
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}$
  $\boldsymbol{D} \leftarrow$ COMPUTED($\boldsymbol{u}$)
  $\boldsymbol{\epsilon'} \leftarrow$ COMPUTESTRAINDEVIATION($\boldsymbol{\epsilon}$)
  for all free cell center $(\mathrm{i,j,k})$ **do**
    $fNorm \leftarrow [(\epsilon'^{\mathrm{i,j,k}}_{xx})^2+(\epsilon'^{\mathrm{i,j,k}}_{yy})^2+(\epsilon'^{\mathrm{i,j,k}}_{zz})^2+2(\epsilon'_{xy}(\boldsymbol{x}^{\mathrm{i,j,k}}))^2+2(\epsilon'_{xz}(\boldsymbol{x}^{\mathrm{i,j,k}}))^2+2(\epsilon'_{yz}(\boldsymbol{x}^{\mathrm{i,j,k}}))^2]^{\frac{1}{2}}$
    $\hat{\epsilon}^{\mathrm{i,j,k}}_{xx} \leftarrow \delta t(D^{\mathrm{i,j,k}}_{xx} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i,j,k}}_{xx}\max(0, fNorm-\lambda))$
    $\hat{\epsilon}^{\mathrm{i,j,k}}_{yy} \leftarrow \delta t(D^{\mathrm{i,j,k}}_{yy} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i,j,k}}_{yy}\max(0, fNorm-\lambda))$
    $\hat{\epsilon}^{\mathrm{i,j,k}}_{zz} \leftarrow \delta t(D^{\mathrm{i,j,k}}_{zz} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i,j,k}}_{zz}\max(0, fNorm-\lambda))$
  end for
  for all free xy-edge $(\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k})$ **do**
    $fNorm \leftarrow [(\epsilon'_{xx}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}))^2 + (\epsilon'_{yy}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}))^2 + (\epsilon'_{zz}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}))^2$
      $+2(\epsilon'^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}_{xy})^2 + 2(\epsilon'_{xz}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}))^2 + 2(\epsilon'_{yz}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}))^2]^{\frac{1}{2}}$
    $\hat{\epsilon}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}_{xy} \leftarrow \delta t(D^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}_{xy} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}}_{xy}\max(0, fNorm-\lambda))$
  end for
  for all free xz-edge $(\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2})$ **do**
    $fNorm \leftarrow [(\epsilon'_{xx}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}))^2 + (\epsilon'_{yy}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}))^2 + (\epsilon'_{zz}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}))^2$
      $+2(\epsilon'_{xy}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}))^2 + 2(\epsilon'^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}_{xz})^2 + 2(\epsilon'_{yz}(\boldsymbol{x}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}))^2]^{\frac{1}{2}}$
    $\hat{\epsilon}^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}_{xz} \leftarrow \delta t(D^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}_{xz} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i}\text{-}\frac{1}{2},\mathrm{j},\mathrm{k}\text{-}\frac{1}{2}}_{xz}\max(0, fNorm-\lambda))$
  end for
  for all free yz-edge $(\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2})$ **do**
    $fNorm \leftarrow [(\epsilon'_{xx}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}))^2 + (\epsilon'_{yy}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}))^2 + (\epsilon'_{zz}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}))^2$
      $+2(\epsilon'_{xy}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}))^2 + 2(\epsilon'_{xz}(\boldsymbol{x}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}))^2 + 2(\epsilon'^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}_{yz})^2]^{\frac{1}{2}}$
    $\hat{\epsilon}^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}_{yz} \leftarrow \delta t(D^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}_{yz} - \frac{\alpha}{fNorm}\epsilon'^{\mathrm{i},\mathrm{j}\text{-}\frac{1}{2},\mathrm{k}\text{-}\frac{1}{2}}_{yz}\max(0, fNorm-\lambda))$
  end for
  return $\hat{\boldsymbol{\epsilon}}$
end procedure

Algorithm 14 Updating the elastic strain

procedure UPDATESTRAIN($\boldsymbol{\epsilon}$,$\boldsymbol{u}$)
  $\boldsymbol{\epsilon}^{(1)} \leftarrow$ ADVECTSTRAIN($\boldsymbol{\epsilon}, \boldsymbol{u}$)
  $\boldsymbol{\epsilon}^{(2)} \leftarrow$ ROTATESTRAIN($\boldsymbol{\epsilon}^{(1)}, \boldsymbol{u}$)
  $\boldsymbol{\epsilon}^{(3)} \leftarrow$ DECAYSTRAIN($\boldsymbol{\epsilon}^{(2)}, \boldsymbol{u}$)
  return $\boldsymbol{\epsilon}^{(3)}$
end procedure

Algorithm 15 Apply Velocity Boundary Conditions

procedure ApplyVelocityBC($\boldsymbol{u}$)
  ExtrapolateVelocity($\boldsymbol{u}$)
  for all SOLID cell (i,j,k) **do**
    if x-face (i-$\frac{1}{2}$,j,k) is a WALL **then**
      $u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow \bar{u}_x^{\text{i-}\frac{1}{2}\text{,j,k}}$
      if SLIP **then**
        $u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow u_y^{\text{i-1,j-}\frac{1}{2}\text{,k}}$
        $u_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow u_z^{\text{i-1,j,k-}\frac{1}{2}}$
      else
        $u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow -u_y^{\text{i-1,j-}\frac{1}{2}\text{,k}}$
        $u_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow -u_z^{\text{i-1,j,k-}\frac{1}{2}}$
      end if
    end if
    if y-face (i,j-$\frac{1}{2}$,k) is a WALL **then**
      $u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow \bar{u}_y^{\text{i,j-}\frac{1}{2}\text{,k}}$
      if SLIP **then**
        $u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow u_x^{\text{i-}\frac{1}{2}\text{,j-1,k}}$
        $u_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow u_z^{\text{i,j-1,k-}\frac{1}{2}}$
      else
        $u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow -u_x^{\text{i-}\frac{1}{2}\text{,j-1,k}}$
        $u_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow -u_z^{\text{i,j-1,k-}\frac{1}{2}}$
      end if
    end if
    if z-face (i,j,k-$\frac{1}{2}$) is a WALL **then**
      $u_z^{\text{i,j,k-}\frac{1}{2}} \leftarrow \bar{u}_z^{\text{i,j,k-}\frac{1}{2}}$
      if SLIP **then**
        $u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow u_x^{\text{i-}\frac{1}{2}\text{,j,k-1}}$
        $u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow u_y^{\text{i,j-}\frac{1}{2}\text{,k-1}}$
      else
        $u_x^{\text{i-}\frac{1}{2}\text{,j,k}} \leftarrow u_x^{\text{i-}\frac{1}{2}\text{,j,k-1}}$
        $u_y^{\text{i,j-}\frac{1}{2}\text{,k}} \leftarrow u_y^{\text{i,j-}\frac{1}{2}\text{,k-1}}$
      end if
    end if
  end for
end procedure

Algorithm 16 Extrapolate velocity across the fluid surface

procedure ExtrapolateVelocity($\boldsymbol{u}$)
  $bandWidth \leftarrow 5\max(\delta x, \delta y, \delta z)$
  Mark all free $u_x^{\text{i-}\frac{1}{2}\text{,j,k}}$, $u_y^{\text{i,j-}\frac{1}{2}\text{,k}}$ and $u_z^{\text{i,j,k-}\frac{1}{2}}$ KNOWN
  $u_xList, u_yList, u_zList \leftarrow Empty$
  Add all non-free $(u_x^{\text{i-}\frac{1}{2}\text{,j,k}}, \Phi(\boldsymbol{x}^{\text{i-}\frac{1}{2}\text{,j,k}}))$ with $\Phi(\boldsymbol{x}^{\text{i-}\frac{1}{2}\text{,j,k}}) \leq bandWidth$ to u_xList
  Add all non-free $(u_y^{\text{i,j-}\frac{1}{2}\text{,k}}, \Phi(\boldsymbol{x}^{\text{i,j-}\frac{1}{2}\text{,k}}))$ with $\Phi(\boldsymbol{x}^{\text{i,j-}\frac{1}{2}\text{,k}}) \leq bandWidth$ to u_yList
  Add all non-free $(u_z^{\text{i,j,k-}\frac{1}{2}}, \Phi(\boldsymbol{x}^{\text{i,j,k-}\frac{1}{2}}))$ with $\Phi(\boldsymbol{x}^{\text{i,j,k-}\frac{1}{2}}) \leq bandWidth$ to u_zList
  Sort all u_xList, u_yList and u_zList by Φ
  while u_xList is not $Empty$ **do**
    $summedWeight \leftarrow 0$
    $(u_x, \Phi) \leftarrow$ First element in u_xList
    for all KNOWN x-face neighbors $\hat{u}_x$ of u_x **do**
      $u_x \leftarrow u_x + (\Phi - \hat{\Phi})\hat{u}_x$
      $summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
    end for
    $u_x \leftarrow u_x / summedWeight$
    Remove u_x from u_xList and mark it KNOWN
  end while
  while u_yList is not $Empty$ **do**
    $summedWeight \leftarrow 0$
    $(u_y, \Phi) \leftarrow$ First element in u_yList
    for all KNOWN y-face neighbors $\hat{u}_y$ of u_y **do**
      $u_y \leftarrow u_y + (\Phi - \hat{\Phi})\hat{u}_y$
      $summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
    end for
    $u_y \leftarrow u_y / summedWeight$
    Remove u_y from u_yList and mark it KNOWN
  end while
  while u_zList is not $Empty$ **do**
    $summedWeight \leftarrow 0$
    $(u_z, \Phi) \leftarrow$ First element in u_zList
    for all KNOWN z-face neighbors $\hat{u}_z$ of u_z **do**
      $u_z \leftarrow u_z + (\Phi - \hat{\Phi})\hat{u}_z$
      $summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
    end for
    $u_z \leftarrow u_z / summedWeight$
    Remove u_z from u_zList and mark it KNOWN
  end while
end procedure

Algorithm 17 Apply Pressure Boundary Conditions

procedure ApplyPressureBC($\boldsymbol{u}$)
  for all non-free cell (i,j,k) **do**
    $p^{i,j,k} \leftarrow 0$
    $numSummed \leftarrow 0$
    if x-face $(i-\frac{1}{2},j,k)$ is a WALL **then**
      $p^{i,j,k} \leftarrow p^{i,j,k} + p^{i-1,j,k}$
      $numSummed \leftarrow numSummed + 1$
    end if
    if y-face $(i,j-\frac{1}{2},k)$ is a WALL **then**
      $p^{i,j,k} \leftarrow p^{i,j,k} + p^{i,j-1,k}$
      $numSummed \leftarrow numSummed + 1$
    end if
    if z-face $(i,j,k-\frac{1}{2})$ is a WALL **then**
      $p^{i,j,k} \leftarrow p^{i,j,k} + p^{i,j,k-1}$
      $numSummed \leftarrow numSummed + 1$
    end if
    if $numSummed > 0$ **then**
      $p^{i,j,k} \leftarrow p^{i,j,k}/numSummed$
    end if
  end for
end procedure

Algorithm 18 Apply Strain Boundary Conditions

procedure ApplyStrainBC($\boldsymbol{\epsilon}$)
  ExtrapolateStrain($\boldsymbol{\epsilon}$)
  for all non-free cell (i,j,k) **do**
    $\epsilon_{xx}^{\text{i,j,k}} \leftarrow 0$
    $\epsilon_{yy}^{\text{i,j,k}} \leftarrow 0$
    $\epsilon_{zz}^{\text{i,j,k}} \leftarrow 0$
    if x-face $(\text{i-}\frac{1}{2}\text{,j,k})$ is a WALL **then**
      $\epsilon_{xx}^{\text{i,j,k}} \leftarrow \epsilon_{xx}^{\text{i,j,k}} + \epsilon_{xx}^{\text{i-1,j,k}}$
      $\epsilon_{yy}^{\text{i,j,k}} \leftarrow \epsilon_{yy}^{\text{i,j,k}} + \epsilon_{yy}^{\text{i-1,j,k}}$
      $\epsilon_{zz}^{\text{i,j,k}} \leftarrow \epsilon_{zz}^{\text{i,j,k}} + \epsilon_{zz}^{\text{i-1,j,k}}$
      $numSummed \leftarrow numSummed + 1$
      $\epsilon_{xy}^{\text{i-}\frac{1}{2}\text{,j-}\frac{1}{2}\text{,k}} \leftarrow 0$
      $\epsilon_{xz}^{\text{i-}\frac{1}{2}\text{,j,k-}\frac{1}{2}} \leftarrow 0$
    end if
    if y-face $(\text{i,j-}\frac{1}{2}\text{,k})$ is a WALL **then**
      $\epsilon_{xx}^{\text{i,j,k}} \leftarrow \epsilon_{xx}^{\text{i,j,k}} + \epsilon_{xx}^{\text{i,j-1,k}}$
      $\epsilon_{yy}^{\text{i,j,k}} \leftarrow \epsilon_{yy}^{\text{i,j,k}} + \epsilon_{yy}^{\text{i,j-1,k}}$
      $\epsilon_{zz}^{\text{i,j,k}} \leftarrow \epsilon_{zz}^{\text{i,j,k}} + \epsilon_{zz}^{\text{i,j-1,k}}$
      $numSummed \leftarrow numSummed + 1$
      $\epsilon_{xy}^{\text{i-}\frac{1}{2}\text{,j-}\frac{1}{2}\text{,k}} \leftarrow 0$
      $\epsilon_{yz}^{\text{i,j-}\frac{1}{2}\text{,k-}\frac{1}{2}} \leftarrow 0$
    end if
    if z-face $(\text{i,j,k-}\frac{1}{2})$ is a WALL **then**
      $\epsilon_{xx}^{\text{i,j,k}} \leftarrow \epsilon_{xx}^{\text{i,j,k}} + \epsilon_{xx}^{\text{i,j,k-1}}$
      $\epsilon_{yy}^{\text{i,j,k}} \leftarrow \epsilon_{yy}^{\text{i,j,k}} + \epsilon_{yy}^{\text{i,j,k-1}}$
      $\epsilon_{zz}^{\text{i,j,k}} \leftarrow \epsilon_{zz}^{\text{i,j,k}} + \epsilon_{zz}^{\text{i,j,k-1}}$
      $numSummed \leftarrow numSummed + 1$
      $\epsilon_{xz}^{\text{i-}\frac{1}{2}\text{,j,k-}\frac{1}{2}} \leftarrow 0$
      $\epsilon_{yz}^{\text{i,j-}\frac{1}{2}\text{,k-}\frac{1}{2}} \leftarrow 0$
    end if
    if $numSummed > 0$ **then**
      $\epsilon_{xx}^{\text{i,j,k}} \leftarrow \epsilon_{xx}^{\text{i,j,k}} / numSummed$
      $\epsilon_{yy}^{\text{i,j,k}} \leftarrow \epsilon_{yy}^{\text{i,j,k}} / numSummed$
      $\epsilon_{zz}^{\text{i,j,k}} \leftarrow \epsilon_{zz}^{\text{i,j,k}} / numSummed$
    end if
  end for
end procedure

Algorithm 19 Extrapolate strain tensor across the fluid surface

procedure EXTRAPOLATESTRAIN($\boldsymbol{\epsilon}$)
$bandWidth \leftarrow 5 \max\{\delta x, \delta y, \delta z\}$
Mark all free $\epsilon_{xx}^{\mathrm{i,j,k}}, \epsilon_{yy}^{\mathrm{i,j,k}}, \epsilon_{zz}^{\mathrm{i,j,k}}, \epsilon_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}, \epsilon_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}, \epsilon_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}$ KNOWN
$\epsilon_{xx}List, \epsilon_{yy}List, \epsilon_{zz}List, \epsilon_{xy}List, \epsilon_{xz}List, \epsilon_{yz}List, \leftarrow Empty$
Add all non-free $(\epsilon_{xx}^{\mathrm{i,j,k}}, \Phi(\boldsymbol{x}^{\mathrm{i,j,k}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i,j,k}}) \leq bandWidth$ to $\epsilon_{xx}List$
Add all non-free $(\epsilon_{yy}^{\mathrm{i,j,k}}, \Phi(\boldsymbol{x}^{\mathrm{i,j,k}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i,j,k}}) \leq bandWidth$ to $\epsilon_{yy}List$
Add all non-free $(\epsilon_{zz}^{\mathrm{i,j,k}}, \Phi(\boldsymbol{x}^{\mathrm{i,j,k}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i,j,k}}) \leq bandWidth$ to $\epsilon_{zz}List$
Add all non-free $(\epsilon_{xy}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}, \Phi(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j\text{-}\frac{1}{2},k}}) \leq bandWidth$ to $\epsilon_{xy}List$
Add all non-free $(\epsilon_{xz}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}, \Phi(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i\text{-}\frac{1}{2},j,k\text{-}\frac{1}{2}}}) \leq bandWidth$ to $\epsilon_{xz}List$
Add all non-free $(\epsilon_{yz}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}, \Phi(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}))$ with $\Phi(\boldsymbol{x}^{\mathrm{i,j\text{-}\frac{1}{2},k\text{-}\frac{1}{2}}}) \leq bandWidth$ to $\epsilon_{yz}List$
Sort all $\epsilon_{xx}List$, $\epsilon_{yy}List$, $\epsilon_{zz}List$, $\epsilon_{xy}List$, $\epsilon_{xz}List$ and $\epsilon_{yz}List$ by Φ
while $\epsilon_{xx}List$ is not $Empty$ **do**
$(\epsilon_{xx}, \Phi) \leftarrow$ First element in $\epsilon_{xx}List$
$(\epsilon_{yy}, \Phi) \leftarrow$ First element in $\epsilon_{yy}List$
$(\epsilon_{zz}, \Phi) \leftarrow$ First element in $\epsilon_{zz}List$
$summedWeight \leftarrow 0$
for all KNOWN cell center neighbors $\hat{\epsilon}_{xx}$ of ϵ_{xx}, $\hat{\epsilon}_{yy}$ of ϵ_{yy} and $\hat{\epsilon}_{zz}$ of ϵ_{zz} **do**
$\epsilon_{xx} \leftarrow \epsilon_{xx} + (\Phi - \hat{\Phi})\hat{\epsilon}_{xx}$
$\epsilon_{yy} \leftarrow \epsilon_{yy} + (\Phi - \hat{\Phi})\hat{\epsilon}_{yy}$
$\epsilon_{zz} \leftarrow \epsilon_{zz} + (\Phi - \hat{\Phi})\hat{\epsilon}_{zz}$
$summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
end for
$\epsilon_{xx} \leftarrow \epsilon_{xx}/summedWeight$
$\epsilon_{yy} \leftarrow \epsilon_{yy}/summedWeight$
$\epsilon_{zz} \leftarrow \epsilon_{zz}/summedWeight$
Remove ϵ_{xx} from $\epsilon_{xx}List$ and mark it KNOWN
Remove ϵ_{yy} from $\epsilon_{yy}List$ and mark it KNOWN
Remove ϵ_{zz} from $\epsilon_{zz}List$ and mark it KNOWN
end while

Algorithm 19 Extrapolate strain tensor across the fluid surface (contin'd)

while $\epsilon_{xy}List$ is not *Empty* **do**
$summedWeight \leftarrow 0$
$(\epsilon_{xy}, \Phi) \leftarrow$ First element in $\epsilon_{xy}List$
for all KNOWN edge neighbors $\hat{\epsilon}_{xy}$ of ϵ_{xy} **do**
$\epsilon_{xy} \leftarrow \epsilon_{xy} + (\Phi - \hat{\Phi})\hat{\epsilon}_{xy}$
$summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
end for
$\epsilon_{xy} \leftarrow \epsilon_{xy}/summedWeight$
Remove ϵ_{xy} from $\epsilon_{xy}List$ and mark it KNOWN
end while
while $\epsilon_{xz}List$ is not *Empty* **do**
$summedWeight \leftarrow 0$
$(\epsilon_{xz}, \Phi) \leftarrow$ First element in $\epsilon_{xz}List$
for all KNOWN edge neighbors $\hat{\epsilon}_{xz}$ of ϵ_{xz} **do**
$\epsilon_{xz} \leftarrow \epsilon_{xz} + (\Phi - \hat{\Phi})\hat{\epsilon}_{xz}$
$summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
end for
$\epsilon_{xz} \leftarrow \epsilon_{xz}/summedWeight$
Remove ϵ_{xz} from $\epsilon_{xz}List$ and mark it KNOWN
end while
while $\epsilon_{yz}List$ is not *Empty* **do**
$summedWeight \leftarrow 0$
$(\epsilon_{yz}, \Phi) \leftarrow$ First element in $\epsilon_{yz}List$
for all KNOWN edge neighbors $\hat{\epsilon}_{yz}$ of ϵ_{yz} **do**
$\epsilon_{yz} \leftarrow \epsilon_{yz} + (\Phi - \hat{\Phi})\hat{\epsilon}_{yz}$
$summedWeight \leftarrow summedWeight + (\Phi - \hat{\Phi})$
end for
$\epsilon_{yz} \leftarrow \epsilon_{yz}/summedWeight$
Remove ϵ_{yz} from $\epsilon_{yz}List$ and mark it KNOWN
end while
end procedure