EFFICIENT PARTICLE-BASED VISCOUS FLUID SIMULATION WITH VIDEO-GUIDED REAL-TO-VIRTUAL PARAMETER TRANSFER

Tetsuya Takahashi

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2020

Approved by:

Ming C. Lin

Dinesh Manocha

Jan F. Prins

David Adalsteinsson

Byungmoon Kim

ABSTRACT

Tetsuya Takahashi: Efficient Particle-based Viscous Fluid Simulation with Video-Guided Real-to-Virtual Parameter Transfer. (Under the direction of Ming C. Lin)

Viscous fluids, such as honey and molten chocolate, are common materials frequently seen in our daily life. These viscous fluids exhibit characteristic behaviors. Capturing and understanding such dynamics have been required for various applications. Although recent research made advances in simulating the viscous fluid dynamics, still many challenges are left to be addressed. In this dissertation, I present novel techniques to more efficiently and accurately simulate viscous fluid dynamics and propose a parameter identification framework to facilitate the tedious parameter tuning steps for viscous materials.

In fluid simulation, enforcing the incompressibility robustly and efficiently is essential. One known challenge is how to set appropriate boundary conditions for free surfaces and solid boundaries. I propose a new boundary handling approach for an incompressible particle-based solver based on the connectivity analysis for simulation particles. Another challenge is that previously proposed techniques do not scale well. To address this, I propose a new multilevel particle-based solver which constructs the hierarchy of simulation particles. These techniques improve the robustness and efficiency achieving the nearly linear scaling unlike previous approaches.

To simulate characteristic behaviors of viscous fluids, such as coiling and buckling phenomena and adhesion to other materials, it is necessary to develop a specialized solver. I propose a stable and efficient particle-based solver for simulating highly viscous fluids by using implicit integration with the full form of viscosity. To simulate more accurate interactions with solid objects, I propose a new two-way fluid-solid coupling method for viscous fluids via the unified minimization. These approaches also improve the robustness and efficiency while generating rotational and sticky behaviors of viscous fluids.

One important challenge for the physically-based simulation is that it is not obvious how to choose appropriate material parameters to generate our desirable behaviors of simulated materials. I propose a parameter identification framework that helps to tune material parameters for viscous fluids with example video data captured from real world fluid phenomena. This framework identifies viscosity parameters for the

real viscous fluids while estimating the hidden variables for the fluids, and enables the parameter transfer from the real world to virtual environment.

To my family, none of this would have been possible without them.

ACKNOWLEDGMENTS

I would like to thank many people who have made this dissertation possible. First and foremost, I thank my adviser Ming C. Lin, who has supported and helped me to overcome the five years of my Ph.D. life at UNC. I believe, without her support and help, I could not complete my work. In particular, I am very grateful that she appreciated my last work and persuaded me to stick to SIGGRAPH. I also thank valuable helps from my committee members: Dinesh Manocha for helping me to improve my presentation and organizing GAMMA group, Jan F. Prins for discussing techniques for parallelizaion algorithms, David Adalsteinsson for teaching me how to evaluate the accuracy of simulation results, and Byungmoon Kim for helping me to widen my knowledge for geometry processing and giving me precious opportunities for my internships at Adobe. All of their supports significantly helped me to complete this dissertation. I also appreciate a help from my coauthors to polish my works, Issei Fujishiro, Yoshinori Dobashi, and Tomoyuki Nishita. I also thank Roberto Camassa, Richard Mclaughlin, and Robert Hunt for discussions on viscous fluids. I have been lucky to have many friends to talk about research and various things: Weizi Li, Atul Rungta, Shan Yang, Zherong Pan, Auston Baker Sterling, Junbang Liang, Srihari Pratapa, Aniket Bera, Chonhyon Park, Nicolas Morales, Andrew Phillip Best, Sahil Narang, Ernest Cheung, Jae Sung Park, Tanmay Randhavane, Justine Wilson, David Wolinski, Pavel Krajcevski, Sujeong Kim, Liang He, Zhenyu Tang, Biao Jia, Tao Yang, and other members in GAMMA group. There are also many friends who helped me to overcome my Ph.D. life at UNC: Aidos Abzhanov, Hasan Faik Alan, Young-Woon Cha, Akash Bapat, True Price, Yipin Zhou, Licheng Yu, Tanya Amert, Eunbyung Park, Abdul Rafay Khalid, Kishore Rathinavel. I also got inspirations from people I met during conferences and internships: Jan Bender, Matthias Teschner, Chris Wojtan, Nils Thuerey, Barbara Solenthaler, Matthias Müller, Miles Macklin, Ryoich Ando, Christopher Batty, Theodore Kim, Yonghao Yue, Dan Koschier, Kiwon Um, Nobuyuki Umetani, Kenshi Takayama, Tsukasa Fukusato, Daichi Ito, Li Yi, I-Chao Shen. I also appreciate a deep support from my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Our world is surrounded by numerous fluid materials. From honey poured onto a pancake, paint used to color walls, to mechanical oils used as a lubricant, we see many different types of fluid materials in various aspects of our everyday life. These complex and fascinating behaviors have captivated many researchers, including mathematicians, physicists, computer scientists, artists, and many others. Significant efforts have been devoted to analyze, understand, and simulate the dynamics of fluids over the decades.

In computer graphics, our primary focus is on how to reproduce or simulate fluids on a computer, typically with specific needs, e.g., fluid behaviors as close as possible to real world counterparts or fluid effects intentionally emphasized for computer animation purposes. There are various ways to generate fluid effects, e.g., procedural animations. However, manually manipulating the fluids to generate sequence of configurations is not practical, unlike rigid body or human animations, because of the extremely high degrees of freedoms for fluids. Thus, it is common to rely on approaches that can automate the sequence generations, e.g., physically-based simulation. To generate fluid effects using physically-based simulation, we typically model the fluids based on the Navier-Stokes equations:

$$\rho \frac{d\mathbf{u}}{dt} = -\nabla p + \nabla \cdot \mathbf{s} + \frac{\rho}{m}\mathbf{F}^{\text{ext}}, \tag{1.1}$$

$$\mathbf{s} = \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right), \tag{1.2}$$

where ρ denotes fluid density, t time, $\mathbf{u}$ velocity, p pressure, $\mathbf{s}$ viscous stress tensor, m mass, $\mathbf{F}^{\text{ext}}$ external force, and μ dynamic viscosity. The Navier-Stokes equations describe how fluid velocity changes preserving the momentum. For simulating fluid dynamics, this physically-based simulation approach has been used in various applications, such as video games, virtual reality, medical training, robotics, and mechanical engineering, and is considered as a standard way for generating fluid animation in the literature.

1.1 Fluid Simulation

To numerically simulate the dynamics of fluids based on the Navier-Stokes equations, it is necessary to discretize these equations. As for the temporal discretization, we typically use implicit integration for robustness and otherwise explicit integration for simplicity. For the spatial discretization, in the computer graphics literature on fluid simulation, there are two commonly used approaches: Eulerian discretization and Lagrangian discretization. These discretization differences characterize the simulation methodologies, and there are advantages and disadvantages over the other.

1.1.1 Eulerian Approach

With the Eulerian approach, simulation domain is discretized as a grid, and physical variables are statically defined at a point on the grid. Then, following the Navier-Stokes equations, we forward the simulation, updating the velocity variables representing the fluid flow.

One important advantage of the Eulerian approach is that we can consistently evaluate the spatial derivatives, e.g., using the staggered grid. Because of the static grid points, we can accurately and efficiently evaluate the spatial derivatives, i.e., gradient, divergence, and Laplacian. This fact makes the results computed with the Eulerian approach consistent and accurate.

Another important advantage for the Eulerian approach lies in enforcing the incompressibility of fluids. In fluid simulation, it is necessary to preserve the volume of fluids for realistic behaviors. In the Eulerian approach, this can be formulated with the divergence free condition:

$$\nabla \cdot \mathbf{u} = 0. \tag{1.3}$$

With this condition, we arrive at the pressure Poisson equation, and then apply pressure gradient after the pressure solve to obtain the divergence-free velocity fields. This procedure can be done by solving a linear system, and is simpler and in general more efficient than the counterpart for the Lagrangian approach.

1.1.2 Lagrangian Approach

With the Lagrangian approach, fluid volumes are discretized with a set of particles, and these particles store the physical quantities. To forward the fluid simulation, these particles are advected with their physical variables following the Navier-Stokes equations.

One important advantage of the Lagrangian approach over the Eulerian approach is that since fluids are directly discretized with a set of particles, the particles need to exist only for places where fluids exist unlike the Eulerian approach which requires the entire simulation domain to be discretized. Because of this advantage, the computational cost and memory usage for the Lagrangian approach can be significantly smaller than those for the grid-based approach.

While the Lagrangian approach can be attractive in terms of the computational cost and memory usage, this approach involves some complexities due to the irregularly distributed particles. One known challenge is that since the evaluations of the spatial derivatives must be performed on moving particles, results can be less accurate and inconsistent. Because of these issues, it is not straightforward to enforce the fluid incompressibility, and various specialized approaches have been actively proposed while still how to handle these problems remains a computational challenge.

1.2 Enforcing Fluid Incompressibility

To simulate realistic fluid behaviors, one of the most important components is to enforce the incompressibility of fluids. Since most liquid materials, such as water, are visually incompressible, we observe that materials that changes their volumes are unrealistic. In the literature, because of the two distinct, Eulerian and Lagrangian discretization, approaches to enforcing the fluid incompressibility differ from each other. With the Eulerian discretization, the most popular approach is to enforce the divergence-free velocity fields following the continuity equation. With this approach, we can formulate a pressure Poisson equation and enforce the divergence-free velocity fields by applying the negative pressure gradient to the velocity fields after the pressure solve. As done in various works, this is relatively straightforward to enforce the incompressibility because of the regular grid structures. On the other hand, with the Lagrangian approach, fluids are typically discretized with a set of particles, and how to enforce the incompressibility for such irregularly distributed particles has been a challenging problem over the decades.

To efficiently enforce the incompressibility of fluids for particle-based methods, various methods have been proposed, e.g., (Koshizuka et al., 1996; Cummins and Rudman, 1999; Premoze et al., 2003; Shao and Lo, 2003; Solenthaler and Pajarola, 2009; He et al., 2012a; Bodin et al., 2012; Macklin and Müller, 2013; Ihmsen et al., 2014a; Kang and Sagong, 2014; Bender and Koschier, 2016). Among these methods, some researchers found that solving a pressure Poisson equation on the particles can be also effective, as done in (Koshizuka et al., 1996; Cummins and Rudman, 1999; Shao and Lo, 2003; Premoze et al., 2003; He et al., 2012a; Ihmsen et al., 2014a). While it is common that this approach requires solving a sparse linear system, variations in discretization of the Laplacian operator lead to the different sparsity of the system and pressure solve. Incompressible SPH (ISPH) (Cummins and Rudman, 1999; Shao and Lo, 2003) and Moving Particle Semi-implicit (MPS) (Koshizuka et al., 1996; Premoze et al., 2003) directly discretize the Laplacian, while a current state-of-the-art particle-based solver, Implicit Incompressible SPH (IISPH) (Ihmsen et al., 2014a) decomposes the Laplacian into divergence and gradient, and then discretizes these operators applying SPH formulations to both operators. When a (weighted) Jacobi method is used, IISPH outperforms ISPH in convergence rate and computational efficiency (Ihmsen et al., 2014a) since IISPH can propagate updated pressures faster by including farther particles with the decomposed operators. However, for ISPH and MPS, we can use a more efficient Conjugate Gradient (CG) method, which is a Krylov subspace method that generally shows faster convergence than stationary iterative methods (e.g., Jacobi and Gauss-Seidel methods), and thus ISPH and MPS can possibly offer performance advantages over IISPH by employing CG. However, ISPH and MPS unfortunately suffer from several stability issues, which make them undesirable for fluid simulation.

When solving the pressure Poisson equation, previously proposed ISPH and MPS methods suffer from experience numerical instabilities because of inaccurately set Dirichlet boundary condition for free surfaces and Neumann boundary condition for solid boundaries. Consequently, to ensure the stable simulations, these methods require smaller time steps, negatively affecting the performance gain due to the fast CG solver. Thus, how to appropriately define boundary conditions for these particle-based solvers remains as a computational challenge.

While employing CG in the pressure solve improves the efficiency significantly over the Jacobi method, several researchers pointed out that CG does not scale well, and the computational cost increases superlinearly with respect to the simulation resolution. In computer graphics, since we generally prefer to generate highly detailed fluid effects within the computation budgets, this superlinearly is undesirable.

In the Eulerian grid-based approach, scalable multigrid methods have been used to address this problem taking advantage of the regular Cartesian grid, which can be used to construct the hierarchy (McAdams et al., 2010; Chentanez and Müller, 2011; Chentanez and Müller, 2012; Ferstl et al., 2014; Weber et al., 2015). This multilevel approach is also adopted for deformable body simulation by constructing the hierarchy from embedded grid structures (Zhu et al., 2010; McAdams et al., 2011) or computing nearly optimal coarser-level structures from finer-levels (Müller, 2008; Wang et al., 2010). Unlike deformable objects, however, it is difficult to apply the multilevel approach to particle-based methods because the connectivity of particles changes at every simulation step, requiring expensive remeshing to keep the quality of coarser-level meshes and to avoid mesh tangling. Thus, how to use the concept of the multilevel solver within the particle-based fluid simulation has not yet been addressed before.

Since how to efficiently and scalably enforce the fluid incompressibility has been a challenging problem for particle-based fluids, this dissertation proposes novel algorithms to address these issues, and demonstrate the effectiveness of the proposed algorithms.

1.3 Simulating Viscous Fluids

Because of the increasing demand for simulating fluids in various applications, approaches to simulating liquids have been developed over the years. In particular, methodologies for simulating incompressible inviscid liquids have been mainly investigated and adopted in practical applications for visual effects and virtual reality. Because of the recent progress in liquid simulations, researchers also started focusing on different types of materials, such as viscous fluids. In general, these fluids have properties different from inviscid fluids, and simulation techniques developed for inviscid fluids are not necessarily applicable. As such, how to effectively simulate viscous fluids remains a major challenge in computer graphics.

One major challenge is that the behaviors of viscous fluids are significantly different from those of inviscid fluids. For example, viscous fluids exhibit damped motions and generate rotational behaviors, such as coiling and buckling phenomena, while inviscid fluids dynamically flow making some splashes. Thus, it is necessary to develop a simulation method for capturing such behaviors unique to viscous fluids. Previously, although some researchers have attempted to simulate viscous fluids using particle-based methods, their approaches typically compromised one of the two aspects: capability or robustness. Some works achieved interesting coiling and buckling effects using an accurate viscosity model, whereas these works use explicit

integration because of the simplicity, and thus suffering from the numerical stability issues. By contrast, other works focus on the numerical stability adopting implicit integration while giving up the capability for the rotational viscous fluid behaviors. Thus, it is required to satisfy both capability and robustness requirements for a practical viscous fluid simulator.

Another important problem on capturing the behaviors of viscous fluids arises in the interactions between viscous fluids and solid objects. In the recent works, several compelling viscous fluid behaviors have been demonstrated focusing primarily on simulating the intriguing behaviors of viscous fluids induced by the gravitational forces with static or prescribed solid boundaries. However, mutual interactions between viscous fluids and solid objects gather little attentions. In the literature, there are various approaches presented to simulate the interactions of inviscid fluids and solids objects, and these approaches achieve the two-way interactions between fluids and solid objects with pressure forces, which play roles of drag and buoyancy forces. The pressure forces can be sufficient to simulate the interactions between nearly inviscid fluids and solid objects. However, since viscous fluids can exert viscosity forces to solid objects, simulating the interactions between highly viscous fluids and solid objects with only pressure forces leads to fluid and solid behaviors significantly different from those observed in the real world, and it is necessary to consider viscosity forces to correctly account for the two-way interactions. In the previous works, how to incorporate the viscosity forces into the fluid simulation has not yet investigated, and thus simulating the two-way interactions between viscous fluids and solid objects has been one important challenge.

To generate plausible and convincing fluid behaviors, physically-based simulation approaches are indispensable because the physically-based simulation can automatically generate the sequence of fluid configurations with appropriately set simulation conditions, such as initial fluid configurations, solid boundaries, simulation parameters, and material parameters. Because of this advantage, physically-based approaches have been extensively used to simulate fluid effects, which are nearly impossible to sufficiently describe with manual approaches due to the very high degrees of freedoms for fluids. However, while physically-based approaches can effectively simulate viscous fluids based on the physical properties of fluids, one known challenge is that it is not clear how to choose appropriate material parameters. It can be very difficult, time-consuming, and tedious to choose appropriate parameters to generate desirable fluid behaviors, e.g., approximating behaviors of viscous fluids observed in the real world. If physical parameters are not ideal, these approaches would generate visually disconcerting results, which negatively impact our sense and recognition of the fluid materials and dampen our experience in various applications, such as video games, movies, and

virtual reality. Even worse, such parameters would cause simulation failure leading to unpredictable results. Consequently, it is necessary to manually tune parameters through tedious trial-and-error processes until we obtain satisfactory visual results. In practice, fluid simulation can take several hours or more, requiring many hours of waiting time to check intermediate results. Thus, such manual parameter-tuning is beyond practical. In addition, from a viewpoint of artists, physical parameters are not necessarily intuitive enough to generate their conceived fluid effects because changes in physical parameters modify fluid flows in a complex and unpredictable way, and the same material parameters can lead to different behaviors depending on simulation scales.

To address these challenges, this dissertation presents several novel algorithms for improving the efficiency and capability of viscous fluid simulators, and propose a parameter identification framework based on real-world captured videos to facilitate the parameter tuning.

1.4 Thesis Statement

My thesis statement is as follows:

Complex dynamics of viscous fluids and their interactions with solid objects can be efficiently and scalably simulated by exploiting regularizers, multilevel representations, and variational principles, and can be effectively reproduced via data-driven simulation with real-world examples.

To support this thesis, I present five methods: a boundary handling method for incompressible particle-based solvers, a multilevel particle-based solver, an implicit viscosity formulation for particle-based methods, a two-way fluid-solid coupling method for viscous fluids, and a video-guided parameter identification framework for real-to-virtual parameter transfer.

1.5 Main Results

The thesis presents the five applications to support each component of the proposed approaches. I list the main results obtained within these applications below.

1.5.1 Boundary Handling for Incompressible Particle-based Solvers

Enforcing the incompressibility of fluids for particle-based methods has been a computational challenge because of the irregular structures of simulation particles. In the literature, a simple Jacobi solver has been

adopted to address the incompressibility problem. However, because of the slow convergence of the Jacobi solver, the computational cost is expensive. Although there is a better solver, such as Conjugate Gradient (CG), to enforce the incompressibility, CG was not applicable to particle-based methods because of the lack of effective boundary handling methods.

In Chapter 2, I propose a boundary handling method for particle-based solver based on incompressible SPH methods to robustly and efficiently simulate incompressible fluids. The main contributions of this work include:

- A free surface boundary handling that appropriately classifies particle roles in the pressure solve based on the particle connectivity analysis, and thus improves the robustness without introducing artifacts.
- A solid boundary handling that introduces a new regularizer term to ensure the solvability of the linear system regardless of the particle configuration. This scheme further improves the robustness and accuracy while enabling the simulation of solid objects floating in the air.
- Kernel blending. This scheme blends densities computed with a smooth kernel and a spiky kernel to improve the robustness of the solver.

By integrating these techniques with the incompressible SPH solver, my method outperforms one of the state-of-the-art fluid solver, IISPH by a factor of 3.78. These results were published in (Takahashi et al., 2016).

1.5.2 Multilevel Particle-based Solver

Enforcing the incompressibility for particle-based fluids can be computationally expensive. This becomes more noticeable when we simulate fluids at a high-resolution since the computational cost for solving incompressibility constraints increases superlinearly. For linear scalability, multigrid methods have been adopted for grid-based simulations. However, how to extend the multigrid methods has been a computational challenge.

In Chapter 3, I proposed a geometric multilevel solver for efficiently solving linear systems arising from particle-based methods. The main contributions of this work include:

- A hierarchy construction method that uses simulation particles and auxiliary grids, and establishes the correspondence between solutions at the particle and grid levels.

- A coarsening scheme that takes simulation elements into account and ensures the solvability of the linear system at coarser levels.
- A solid boundary handling that can be unified with a pressure Poisson equation.

By combining these schemes, my method achieves the nearly linear scaling with respect to the particle resolutions, and outperforms the state-of-the-art fluid solver, IISPH, by a factor of up to 10.0. These results were published in (Takahashi and Lin, 2016).

1.5.3 Implicit Viscosity Formulation for Particle-based Methods

Unlike inviscid fluids, viscous fluids exhibit characteristic behaviors, such as coiling and buckling phenomena. To faithfully simulate such viscous fluid behaviors, it is necessary to develop a specialized technique for this purpose. In the literature, however, researchers have relied on either a simplified viscosity model, which cannot capture rotational behaviors of viscous materials, or explicit time integration with the full form of viscosity, compromising the robustness of the solver. Thus, there is no effective particle-based solvers for robustly and accurately simulating highly viscous fluids.

In Chapter 4, I propose a new formulation with the full form of viscosity and implicit integration to robustly and efficiently simulate viscous fluids without compromising the capability for the rotational viscous fluid behaviors. The main contributions of this work include:

- A new implicit viscosity formulation that is based on the variational principle and improves the robustness and efficiency while allowing for rotational viscous fluid motions.
- A coefficient extraction method that assembles the entries of linear system by accessing the neighbor particles' neighbor particles.

By combining these techniques, my method achieves 3.4x performance gain over previous methods that use explicit integration while achieving the characteristic coiling and buckling behaviors. These results were published in (Takahashi et al., 2015).

1.5.4 Two-Way Fluid-Solid Coupling Method for Viscous Fluids

While various approaches have been proposed to simulate viscous fluids due to the high demand in a wide range of applications, few research has been conducted to simulate how viscous fluids interact with

solid objects. Although some researchers proposed two-way fluid-solid coupling method via pressure forces, considering only pressure forces is insufficient for highly viscous fluids, which can exert viscosity forces to solid objects.

In Chapter 5, I propose a grid-based fluid solver for simulating viscous materials and their interactions with solid objects. The main contributions of this work include:

- A geometrically consistent volume fraction estimation that utilizes the supersampling to improve the accuracy while avoiding the dangling artifacts near free surfaces and solid objects with very little additional cost.
- A two-way fluid-solid coupling method that robustly and correctly accounts for the dynamics of both viscous fluids and solid objects simultaneously through their interactions.
- A position-correction method that uses density constraints to enforce the uniform distributions of particles to avoid non-physical volume changes.

I integrate these techniques with a viscous fluid solver and achieve up to 10% relative errors, as compared to the analytical solution in the ball falling test, which is several orders of magnitude more accurate than previous methods. These results were published in (Takahashi and Lin, 2019a).

1.5.5 Video-Guided Real-to-Virtual Parameter Transfer

In physically-based simulation, it is essential to choose appropriate material parameters to generate desirable simulation results. In many cases, however, appropriately choosing material parameters is very challenging, and often tedious trial-and- error parameter tuning steps are inevitable. Since fluid simulation would take a couple of hours to check the intermediate results, requiring us to wait, the manual parameter tuning is beyond practical.

In Chapter 6, I propose a real-to-virtual parameter transfer framework for facilitating the parameter identification of viscous fluids with example video data captured from real-world phenomena. The main contributions of this work include:

- A parameter optimization framework that identifies the viscosity parameters for fluids based on example videos captured from real-world fluid phenomena, simultaneously inferring hidden physical quantities of fluids.

- Screen space evaluation that allows for measuring differences between the example data and simulation results without reconstructing 3D data.
- Parameter transfer from real to virtual environments. My work introduces a new data-driven approach for fluid animation and enables us to reproduce fluid behaviors in the virtual environment, preserving the observed fluid properties in the real world.

I present a parameter identification framework for fluids based on guiding examples, videos captured from real world fluid phenomena, and demonstrated that the identified parameters can be used in novel scenarios, achieving the real-to-virtual parameter transfer. These results were published in (Takahashi and Lin, 2019b).

1.6 Organization

The remainder of this dissertation is organized as follows. The discussion on the boundary handling method for particle-based methods is given in Chapter 2. This is followed by my work on the scalable multilevel particle-based solver in Chapter 3. Then, I discuss an implicit viscosity formulation for particle-based methods in Chapter 4, and explain how to improve the accuracy of two-way fluid-solid coupling for hybrid Eulerian-Lagrangian methods in Chapter 5. Finally, I present a parameter tuning framework for fluid materials based on video guiding in Chapter 6. Chapter 7 concludes this dissertation by presenting a summary of these works, their contributions, and discussions of future work.

CHAPTER 2: An Efficient Hybrid Incompressible SPH Solver with Interface Handling for Boundary Conditions

2.1 Introduction

Particle-based methods, such as Smoothed Particle Hydrodynamics (SPH), have been widely used to generate visual effects of fluids in computer graphics due to advantages of Lagrangian representations (Ihmsen et al., 2014b; Macklin et al., 2014). In these methods, however, enforcing fluid incompressibility has been a computational challenge, and various methods have been proposed to address this problem (Koshizuka et al., 1996; Cummins and Rudman, 1999; Premoze et al., 2003; Shao and Lo, 2003; Solenthaler and Pajarola, 2009; He et al., 2012a; Bodin et al., 2012; Macklin and Müller, 2013; Ihmsen et al., 2014a; Kang and Sagong, 2014; Bender and Koschier, 2016). Among these methods, solving a Pressure Poisson Equation (PPE) has been shown to be an effective approach (Koshizuka et al., 1996; Cummins and Rudman, 1999; Shao and Lo, 2003; Premoze et al., 2003; He et al., 2012a; Ihmsen et al., 2014a). While it is common that this approach requires solving a sparse linear system, variations in discretization of the Laplacian operator lead to the different sparsity of the system and pressure solve. Incompressible SPH (ISPH) (Cummins and Rudman, 1999; Shao and Lo, 2003) and Moving Particle Semi-implicit (MPS) (Koshizuka et al., 1996; Premoze et al., 2003) directly discretize the Laplacian, while a current state-of-the-art particle-based solver, Implicit Incompressible SPH (IISPH) (Ihmsen et al., 2014a) decomposes the Laplacian into divergence and gradient,

Figure 2.1: A large scale corridor flood simulated with my solver, where 1.40 M fluid particles and 0.29 M solid particles are used. (Left) opaque mesh view. (Middle) particle view, where cyan, magenta, and yellow particles represent Poisson, Dirichlet, and Neumann particles, respectively (see § 2.4.1). (Right) final rendered view.

and then discretizes these operators applying SPH formulations to both operators. When a (weighted) Jacobi method is used, IISPH outperforms ISPH in convergence rate and computational efficiency (Ihmsen et al., 2014a) since IISPH can propagate updated pressures faster by including farther particles with the decomposed operators. However, for ISPH and MPS, we can use a more efficient Conjugate Gradient (CG) method, which is a Krylov subspace method that generally shows faster convergence than stationary iterative methods (e.g., Jacobi and Gauss-Seidel methods). CG is inapplicable to IISPH, as pointed out in (Ihmsen et al., 2014a), due to the non-positive-definite property of the coefficient matrix (see § 2.6.1 for details). Additionally, the number of Jacobi iterations for IISPH increases super-linearly with time steps making the use of larger time steps ineffective. Because of these issues, ISPH and MPS can possibly offer performance advantages over IISPH by employing CG.

However, ISPH and MPS unfortunately have several issues, which make them undesirable for fluid simulation. First, ISPH uses a smooth kernel to compute the particle density in the source term. The resulting particle density is smooth and does not significantly increase although particles are almost overlapped. Consequently, ISPH with this smooth kernel is more likely to fail to prevent fluid-solid particle penetrations. On the other hand, MPS uses a spiky kernel for the density computation. Thus, as particles approach others, the resulting density rapidly increases effectively preventing fluid-solid particle penetrations. However, because of the rapid density changes, the spiky kernel tends to cause instabilities when fluid-fluid collisions occur. Second, previously proposed free surface handling methods for specifying Dirichlet boundary conditions are likely to undergo numerical instabilities mainly because of rough estimates of physical values. The density-based (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) and surface-based (He et al., 2012b) methods do not take into consideration the predicted density and actual pressures of neighboring particles while the ghost-particle-based method (Nair and Tomar, 2014) relies on very rough estimates and assumptions based on ghost particles. Though the source-term-based method (which depends purely on the source term) considers the predicted density, this method disregards actual pressures of neighbor particles. Third, the previously used solid boundary handling for Neumann boundary conditions (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003), which treats solid particles as fluid ones, results in a much larger system of equations thereby increasing the overall computational cost, and erroneously estimates particle pressures to ensure the solvability of the linear system. Furthermore, even though this approach is used, the system can become unsolvable due to objects floating in the air.

I propose a hybrid SPH solver with a new interface handling method for addressing the three aforementioned problems. The main results of this work are as follows:

- A hybrid solver that blends the particle density computed with a smooth and a spiky kernel to take advantage of the specific properties of these kernels. This improves the robustness for both fluid-fluid and fluid-solid collisions while taking larger time steps and thus leading to more efficient simulation.
- A free surface handling method for Dirichlet boundary conditions that appropriately determines fluid particles by employing Jacobi-based pressure prediction. This takes into account not only the predicted density but also pressures of neighboring particles, to improve the robustness.
- A solid boundary handling method for Neumann boundary conditions, which introduces a new term to ensure the solvability of the linear system. This new term offers three advantages over the previous methods. First, I can completely exclude solid particles treated as unknown variables from the PPE thereby significantly reducing the size of the system and thus computational cost. Second, my solid boundary handling can avoid underestimation of particle pressures enabling the use of larger time steps. Third, I can handle objects floating in the air with the direct Laplacian discretization making it possible to simulate two-way interactions, which frequently cause objects floating in the air because of fluid-solid and solid-solid collisions.

By taking advantage of these, I demonstrate that my solver outperforms other particle-based solvers. Figure 2.1 illustrates a large-scale corridor flood scene involving two-way fluid-solid interactions, simulated with my method.

2.2 Related Work

Many particle-based methods have been proposed in Computational Fluid Dynamics (CFD) and computer graphics. I refer readers to (Monaghan, 2005) for a survey on SPH in CFD and (Ihmsen et al., 2014b) for applications of particle-based methods in computer graphics. Below I focus my discussions on particle-based methods most closely related to mine.

ISPH. ISPH was originally proposed by Cummins and Rudman (Cummins and Rudman, 1999) introducing an idea of the pressure projection (which is commonly used in the Eulerian approach (Bridson, 2008)) into SPH. In this method, the divergence of velocity is used as a source term in the PPE. However, errors to the

divergence-free velocity field can accumulate, leading to volume changes. To address this, Shao and Lo (Shao and Lo, 2003) proposed a new source term using the rest density (known as the density invariance source term), which prevents the error accumulation, also presenting free surface handling for Dirichlet boundary conditions to keep the PPE solvable with the modified source term, which does not satisfy the compatibility condition (Bridson, 2008). In addition, Shao and Lo (Shao and Lo, 2003) replaced a mirroring solid boundary handling method, used in (Cummins and Rudman, 1999), with a method that uses solid boundary particles to handle complex geometry. Nair and Tomar (Nair and Tomar, 2014) proposed a new surface handling method for ISPH, adapting an idea of ghost particles, and ensured solvable linear systems without Dirichlet boundary conditions at the expense of difficult parameter tuning and inaccurately estimated pressures. Cummins and Rudman (Cummins and Rudman, 1999) applied a multigrid approach to the ISPH. However, this approach is imported from the grid-based approach, and cannot handle irregular domains. Additionally, their application is limited to fluid simulation without free surfaces (Dirichlet boundary condition) and complex solid objects (Neumann boundary condition).

MPS. MPS is a particle-based pressure projection method proposed in (Koshizuka et al., 1996; Premoze et al., 2003). In the algorithm level, MPS and ISPH (presented in (Shao and Lo, 2003)) are the same, and the differences lie in two aspects: (1) the (number) density computation for the source term, and (2) discretization for spatial derivatives. To compute the density, MPS uses a spiky kernel whose values rapidly increase when particles are close. This effectively prevents fluid-solid particle penetrations whereas the rapidly increased density tends to cause stability issues with fluid-fluid collision shocks. On the other hand, ISPH uses a smooth kernel, which leads to the smooth density distribution. This is more robust against fluid-fluid collisions whereas ISPH is more likely to fail to prevent fluid-solid particle penetrations. In MPS, since the gradient formulation is not anti-symmetric, and thus pressure forces computed with the gradient do not preserve fluid momentum.

IISPH. IISPH is a recently proposed particle-based method that uses the pressure projection with divergence and gradient operators instead of the Laplacian (Ihmsen et al., 2014a). IISPH includes farther particles to faster propagate pressure updates with weighted Jacobi than ISPH and MPS. Additionally, to improve the robustness, Ihmsen et al. (Ihmsen et al., 2014a) proposed adopting a velocity-based density estimation (which uses the continuity equation with predicted particle velocities) instead of the position-based density estimation (which uses the summation approach with predicted particle positions) adopted in

Table 2.1: Feature comparison for density blending.

Method	Fluid-fluid collision	Fluid-solid collision
Smooth kernel (Müller et al., 2003)	Robust	Weak
Spiky kernel (Koshizuka et al., 1996)	Weak	Robust
My method	Robust	Robust

Table 2.2: Feature comparison for free surface handling.

Method	Robustness
Density-based (Koshizuka et al., 1996)	Weak
Surface-based (He et al., 2012b)	Weak
Ghost-particle-based (Nair and Tomar, 2014)	Weak
Source-term-based	Moderate
My method	Robust

ISPH (Shao and Lo, 2003) and MPS. In IISPH, Ihmsen et al. used a clamping approach with weighted Jacobi for free surface handling and mirrored hydrodynamic forces (Akinci et al., 2012) for solid boundary handling.

There are some techniques for accelerating the pressure solve. Kang and Sagong (Kang and Sagong, 2014) and Bender and Koschier (Bender and Koschier, 2016) adjusted particle velocities after fluid incompressibility is enforced to reduce the deviation of particle density at the next step. Adams et al. (Adams et al., 2007) and Solenthaler and Gross (Solenthaler and Gross, 2011) used adaptive particles to allocate more computational resources to important regions. These techniques are orthogonal and can be combined with my method for better performance.

I combine different advantages of ISPH, MPS, and IISPH to achieve optimal performance with my new interface handling method, which allows us to use CG with direct Laplacian discretization. Therefore, my method outperforms the previous particle-based fluid solvers including variants of ISPH with CG. For clarity, features of previous approaches and my method are summarized in Tables 2.1, 2.2, and 2.3, in terms of the density blending, free surface handling, and solid boundary handling, respectively.

Table 2.3: Feature comparison for solid boundary handling.

Method	System size	Pressure	Solid floating in the air
Previous method (Koshizuka et al., 1996)	Large	Inaccurate	✗
My method	Small	More accurate	✓

2.3 Hybrid Incompressible SPH Solver

I first explain formulations of my hybrid fluid solver, which takes advantages of ISPH, MPS, and IISPH to improve the robustness and efficiency, adopting the fluid-solid coupling method (Akinci et al., 2012). Then, I present my interface handling method in § 2.4.

Incompressible flows in the Lagrangian setting can be described by the continuity equation $\frac{d\rho_i}{dt} + \rho_i \nabla \cdot \mathbf{u}_i = 0$, and the Navier-Stokes equations $\frac{d\mathbf{u}_i}{dt} = -\frac{1}{\rho_i}\nabla p_i + \frac{\mathbf{F}_i^{\mathrm{v}}}{m_i} + \frac{\mathbf{F}_i^{\mathrm{ext}}}{m_i}$, where ρ_i denotes density of particle i, t time, $\mathbf{u}_i$ velocity, p_i pressure, $\mathbf{F}_i^{\mathrm{v}}$ viscosity force, m_i mass, and $\mathbf{F}_i^{\mathrm{ext}}$ external force. First, I compute density ρ_i and number density n_i by using the summation approach with a smooth kernel W_{ij} and a spiky kernel w_{ij}, respectively, with solid particles:

$$\rho_i = \sum_j m_j W_{ij} + \rho_0 \sum_s V_s W_{is}, \tag{2.1}$$

$$n_i = \sum_j w_{ij} + \frac{1}{V_0} \sum_s V_s w_{is}, \tag{2.2}$$

where j and s denote neighbor fluid and solid particles, respectively, ρ_0 the rest density, V_i the volume, and V_0 the rest volume defined as $V_0 = \frac{1}{\sum_j W_{ij}}$ at the initial setting. I use $V_i = \frac{m_i}{\rho_i}$ for the fluid particle volume, while I define the solid particle volume as $V_i = \frac{1}{\sum_s W_{is}}$ (see (Akinci et al., 2012)). Note that I initialize particle mass by $m_i = \frac{\rho_0}{\sum_j W_{ij}}$ and compute the rest number density n_0 by $n_0 = \sum_j w_{ij}$ with the initial particle configuration to enforce $\frac{\rho_i}{\rho_0} = \frac{n_i}{n_0}$ to hold, making density and number density in the different dimensions interchangeable with just scaling while achieving the equilibrium state with the initial setup. I use kernels (including a smooth kernel) proposed in (Müller et al., 2003) for SPH discretization, and a spiky kernel used in (Koshizuka et al., 1996; Premoze et al., 2003).

I estimate intermediate velocity $\mathbf{u}_i^*$ with viscosity and external forces. To take advantage of both of the kernels, I blend ρ_i and n_i with scaling as

$$\tilde{\rho}_i = (1 - \zeta_i)\rho_i + \zeta_i \frac{\rho_0}{n_0} n_i, \tag{2.3}$$

$$\zeta_i = \begin{cases} 0 & M_i < \alpha \\ \frac{M_i - \alpha}{\beta - \alpha} & \alpha \le M_i < \beta \\ 1 & \beta \le M_i \end{cases},$$

where $\tilde{\rho}_i$ denotes blended density, M_i the number of neighbor solid particles, and α and β are tunable parameters (I typically use $\alpha = 5$ and $\beta = 15$). To interpolate ρ_i and n_i in Eq. (2.3), I compute ζ_i such that ρ_i and n_i become dominant inside of the fluid volume and near solids, respectively. Since ρ_i and n_i are more robust against fluid-fluid and fluid-solid collisions, respectively, the blended density with ζ_i takes advantages of both kernels, and is more robust than non-blended densities. Then, I compute intermediate density ρ_i^* using the continuity equation as in (Ihmsen et al., 2014a):

$$\rho_i^* = \tilde{\rho}_i + \Delta t \left(\sum_j m_j \mathbf{u}_{ij}^* \nabla W_{ij} + \rho_0 \sum_s V_s \mathbf{u}_{is}^* \nabla W_{is} \right), \tag{2.4}$$

where Δt denotes time step, and $\mathbf{u}_{ij}^* = \mathbf{u}_i^* - \mathbf{u}_j^*$.

Next, I solve the PPE to obtain a pressure field that enforces fluid incompressibility. Assuming that the density change in the continuity equation is caused by the pressure forces, I formulate the PPE for the fluid domain Ω as $\nabla^2 p_i = \frac{\rho_0 - \rho_i^*}{\Delta t^2}$ in Ω with Dirichlet boundary condition $p_i = 0$ on F and Neumann boundary condition $\frac{dp_i}{d\hat{\mathbf{n}}_i} = 0$ on S, where F and S denote domain boundaries in contact with free surfaces and solid objects, respectively, and $\hat{\mathbf{n}}_i$ is the normal to S. In particle-based methods, negative pressures which can occur because of the positive source term if the PPE is solved without special cares produce attractive forces between particles leading to tensile instability (Monaghan, 2000). Thus, we need to satisfy $p_i \geq 0$ while solving the PPE, and this is a Linear Complementarity Problem (LCP), e.g., solved to avoid particle adhesion (Adams and Wicke, 2009; Narain et al., 2010; Alduán and Otaduy, 2011; Ihmsen et al., 2013; Gerszewski and Bargteil, 2013; Ihmsen et al., 2014a).

After the PPE is solved by using a CG solver, satisfying the boundary conditions and constraint on pressure with my interface handling (see § 2.4), I compute pressure forces, and then integrate particle velocities and positions using the semi-implicit Euler method.

I summarize the algorithm for my hybrid incompressible SPH solver in Algorithm 1. An algorithm for solving the PPE (line 9 in Algorithm 1) is given in Algorithm 2 (§ 2.4.6).

2.4 Interface Handling

In this section, I concisely describe boundary conditions in the PPE (§ 2.4.1). Next, I clarify problems on Dirichlet (§ 2.4.2) and Neumann (§ 2.4.3) boundary conditions in the ISPH and MPS setting, and then explain

Algorithm 1 Hybrid incompressible SPH solver

for all particle i **do**
    find neighbor particles
for all fluid particle i **do**
    compute ρ_i and n_i with Eqs. (2.1) and (2.2)
for all particle i **do**
    compute intermediate velocity $\mathbf{u}_i^*$
for all fluid particle i **do**
    estimate ρ_i^* with Eq. (2.4)
solve the PPE (Algorithm 2)
for all fluid particle i **do**
    compute pressure force $\mathbf{F}^{\mathrm{p}}$
for all fluid particle i **do**
    integrate velocity $\mathbf{u}_i^{t+1}$ and position $\mathbf{x}_i^{t+1}$

my interface handling consisting of free surface handling (§ 2.4.4) and solid boundary handling (§ 2.4.5). Finally, I give an algorithm for solving the PPE (§ 2.4.6).

I use my solid boundary handling method for Neumann boundary conditions in the PPE and employ the work of Akinci et al. (Akinci et al., 2012) to resolve collisions between fluid and solid particles. (Akinci et al., 2012) is also suitable for my free surface handling method since it allows for accurate density estimates even with complex geometry. My free surface handling is for Dirichlet boundary conditions and is orthogonal to other free surface handling methods that consider air domains, e.g., (Schechter and Bridson, 2012; He et al., 2014). Thus, these methods can be combined with mine.

In my experiments, quantities computed with SPH and MPS discretization for the Laplacian operator are almost the same and are virtually interchangeable. However, since the gradient formulation in MPS is not anti-symmetric, I use SPH discretization for consistency to compute the spatial derivatives.

2.4.1 Boundary Conditions in PPE

To aid in my description, I call fluid and solid particles included in the PPE as unknown variables *Poisson particles*, fluid particles used for Dirichlet boundary condition *Dirichlet particles*, and solid particles for Neumann boundary condition *Neumann particles*.

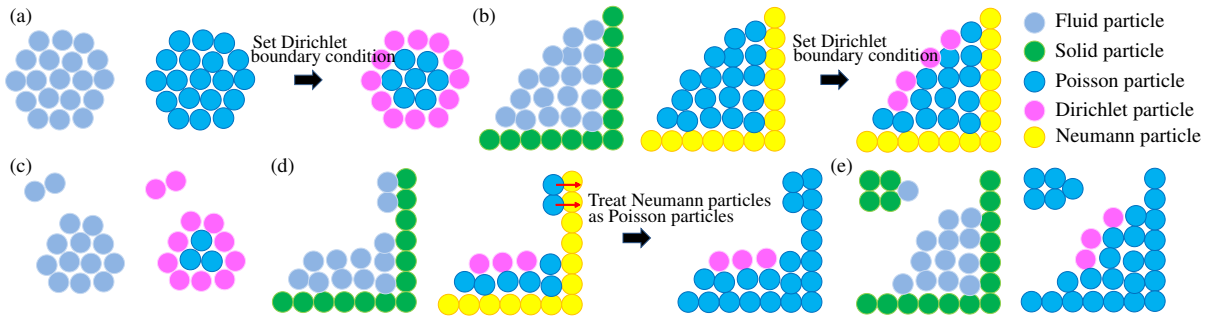


Figure 2.2: Illustration of particle configurations. Red arrows represent particle velocities. (a) Configuration of fluid particles, unsolvable configuration of Poisson particles, and solvable configuration of Poisson and Dirichlet particles due to the newly set Dirichlet boundary condition, from left to right. (b) Configuration of fluid and solid particles, unsolvable configuration of Poisson and Neumann particles, and solvable configuration of Poisson, Dirichlet, and Neumann particles. (c) Configuration of fluid particles, and solvable configuration of Poisson and Dirichlet particles. (d) Configuration of fluid and solid particles, unsolvable configuration of Poisson, Dirichlet, and Neumann particles, and solvable configuration of Poisson and Dirichlet particles due to the new Poisson particles converted from the Neumann particles. (e) Configuration of fluid and solid particles, and unsolvable configuration of Poisson and Dirichlet particles due to the group of isolated Poisson particles without Dirichlet particles.

To solve the PPE, I can discretize $\nabla^2 p_i$ using the SPH formulation taking solid particles into account with the assumption of virtually existing solid particle's pressure p_s:

$$\begin{aligned} \nabla^2 p_i &= \nabla^2 p_i^{\text{fluid}} + \nabla^2 p_i^{\text{solid}} \\ &= \sum_j a_{ij}(p_i - p_j) + \sum_s a_{is}(p_i - p_s), \end{aligned} \tag{2.5}$$

where $a_{ij} = V_{ij}\hat{W}_{ij}$, $V_{ij} = (V_i + V_j)/2$, $\hat{W}_{ij} = 2\frac{\mathbf{x}_{ij}\cdot\nabla W_{ij}}{\|\mathbf{x}_{ij}\|^2 + 0.01h^2}$, $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ ($\mathbf{x}_i$: particle position), and h denotes kernel radius. $a_{ij} < 0$ because of the kernel definition (Müller et al., 2003). For Neumann boundary condition ($\frac{dp_i}{d\mathbf{n}_i} = 0$), $p_i = p_s$ must be satisfied, and thus the PPE becomes

$$\nabla^2 p_i = \sum_j a_{ij}(p_i - p_j) = c_i,$$

where $c_i = \frac{\rho_0 - \rho_i^*}{\Delta t^2}$. Unlike the Eulerian fluid simulation, which uses the divergence of velocity as a source term, the density invariance source term generally does not satisfy the compatibility condition (even if the source term is not blended), i.e., $\sum_{i\in\Omega^{\mathrm{P}}} c_i \neq 0$, where Ω^{P} denotes a set of Poisson particles. Therefore, this form of the PPE without Dirichlet boundary conditions is unsolvable because of the rank deficient coefficient matrix (see (Bridson, 2008) for more details). Figure 2.2 (a) illustrates a particle configuration without solid

particles for an unsolvable setting consisting of Poisson particles only, and a solvable setting due to Dirichlet particles. Figure 2.2 (b) illustrates a similar situation with solid Neumann particles.

2.4.2 Problems on Dirichlet Boundary Condition

Unlike Eulerian fluid simulation, negative pressures almost always work negatively, causing the tensile instability (Monaghan, 2000) in particle-based fluid simulation. Thus, it is essential to avoid negative pressures, and formally, we need to solve an LCP. With (weighted) Jacobi method, this step can be easily achieved by clamping negative pressures to zero in each iteration, whereas CG does not allow us to perform such an operation in its iterations (Ihmsen et al., 2014a). More expensive LCP solvers have been used in the Eulerian methods (Narain et al., 2010; Gerszewski and Bargteil, 2013). By contrast, in Lagrangian particle-based methods, I approximately solve the LCP to avoid using the costly LCP solvers by solving the PPE with Dirichlet boundary conditions, which are set to particles whose pressures should be negative after the PPE is solved (See Figure 2.2). Previously, the density-based (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) or surface-based (He et al., 2012b) method has been used to determine Dirichlet particles. However, since predicted density and pressures of fluid particles are neglected, these methods tend to fail to appropriately determine Dirichlet particles, i.e., fluid particles whose pressures should be positive (negative) after the PPE is solved could be erroneously treated as Dirichlet (Poisson) particles consequently leading to stability issues. The source-term-based method takes the predicted density into account, and thus its stability is improved. However, disregarded neighbor particles' pressures still negatively affect the stability of the simulation.

The ghost-particle-based method (Nair and Tomar, 2014) can change an unsolvable system consisting of Poisson and Neumann particles only (without Dirichlet particles) to a solvable one by setting Dirichlet boundary conditions to ghost particles, assuming that they virtually exist outside of the fluid domain. When the particle uniformity is disturbed, however, the linear system can be not diagonally-dominant, and thus not positive-definite. Consequently, we fail to solve the system using CG. Although taking larger diagonal entries can ensure the diagonal dominance of the linear system, resultant pressures can be much smaller failing to prevent particle penetrations with mirrored hydrodynamic forces (Akinci et al., 2012).

2.4.3 Problems on Neumann Boundary Condition

When Poisson particles have a channel to at least one Dirichlet particle directly or via other Poisson particles, the solvability of the PPE is ensured. In liquid simulations, however, fluid particles can be separated from the fluid bulk that has Dirichlet particles (see Figures 2.2 (c) and (d)). If there is no neighboring solid particle, I can appropriately set Dirichlet boundary conditions with a free surface handling method ensuring the solvability of the system, as illustrated in Figure 2.2 (c). In this case, although pressure forces of Dirichlet particles do not resolve their collisions because of the zero-pressures, artificial viscosity force, which is necessary to stabilize particle behaviors, can prevent fluid particle penetrations. On the other hand, if fluid particles have neighboring solid particles (see Figure 2.2 (d)), I cannot set Dirichlet boundary condition ($p_i = 0$) to the fluid particles except the cases, where fluid particles do not move toward solid particles. This is because pressures of fluid particles must be valid for (Akinci et al., 2012), and I need to treat the fluid particles as Poisson particles. In this case, the resulting linear system is unsolvable because of isolated groups of Poisson particles without Dirichlet particles (see Figure 2.2 (d)). To avoid this unsolvable configuration, I tested pressures computed with an equation of state instead of setting $p_i = 0$ and artificial viscosity excluding Poisson particles that contact with Neumann particles from the system, I could not determine appropriate parameters and experienced significant fluid energy dissipations.

This problem was partially addressed in (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) by treating solid particles as Poisson particles in the PPE (see Figure 2.2 (d)). This approach can connect separated fluid Poisson particles to Dirichlet particles via solid Poisson particles which are originally Neumann particles, and thus can change an unsolvable PPE to a solvable one. However, this approach brings about several other problems. First, solid Poisson particles newly included in the PPE significantly increase the size of the system and thus computational cost and memory usage. Second, this approach tends to underestimate pressures requiring smaller time steps to ensure no fluid-solid particle penetrations because solid Poisson particles newly generate shorter channels between fluid Poisson particles and Dirichlet particles limiting the pressures of the fluid Poisson particles to lower values. Most importantly, this approach cannot handle solid objects floating in the air, which frequently occur in the two-way solid-fluid coupling (see Figure 2.1 and accompanying video), since solid particles of these objects may not have channels to Dirichlet particles (see Figure 2.2 (e)).

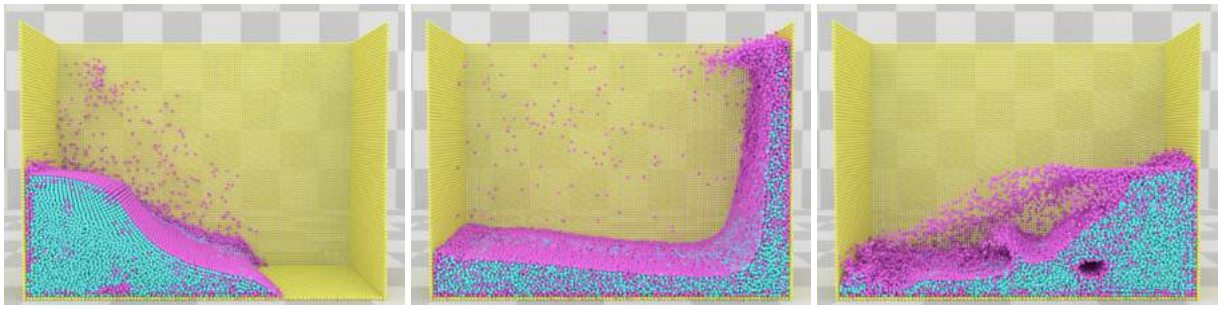


Figure 2.3: Cutaway views for a dam break scene. Dirichlet particles are appropriately set near free surfaces and cavities inside of the fluid.

2.4.4 Free Surface Handling

I aim to approximately solve the LCP by appropriately setting Dirichlet boundary conditions to particles, whose pressures would be negative after the PPE is solved. To estimate particle pressures, I compute a pseudo particle pressure $\tilde{p}_i$ using the projected Jacobi method based on Eq. (2.5):

$$\tilde{p}_i^{l+1} = \max\left(0, \frac{c_i + \sum_j a_{ij}\tilde{p}_j^l + \tilde{p}_i^l \sum_s a_{is}}{\sum_j a_{ij} + \sum_s a_{is}}\right), \tag{2.6}$$

where l denotes iteration count, and the initial pseudo pressures are set as $\tilde{p}_i^0 = \gamma p_i^t$ with a tunable parameter γ. Then, if $\tilde{p}_i^l = 0$, I treat the fluid particle i as a Dirichlet particle, otherwise a Poisson particle. I use a projected Jacobi method, not Jacobi method, since Jacobi method excessively increases the number of Dirichlet particles propagating negative pressures. Note that since my Jacobi-based prediction includes pressures of neighboring particles and the source term, which is based on the predicted density ρ_i^* and thus density ρ_i, I can appropriately set Dirichlet boundary conditions to particles, e.g., on free surfaces and near cavities inside of fluid, as shown in Figure 2.3.

Since repeatedly applying Eq. (2.6) means solving the PPE using the projected Jacobi method as in (Ihmsen et al., 2014a), I can obtain more accurate particle pressures to determine Dirichlet particles with many iterations. However, using many iterations did not significantly improve the stability. Additionally, since all particle pressures are zero or positive, repeatedly applying Eq. (2.6) generally increases pseudo pressures and thus decreases the number of Dirichlet particles. Whereas Dirichlet particles can cause stability issues if they are excessive and inappropriately set, they can reduce the number of Poisson particles in the system improving computational efficiency and memory usage and accelerate the convergence of CG because

of their fixed pressure values and the decreased system size. Taking these into account, I use Eq. (2.6) only once with $\gamma = 0.5$ not to excessively decrease the number of Dirichlet particles.

Since negative pressures can still occur with Dirichlet particles specified using my free surface handling, I clamp negative pressures to zero to prevent the tensile instability (Monaghan, 2000) after the PPE is solved.

It is worth noting that an ad-hoc technique, clamping the positive source term to zero, can completely prevent negative pressures. However, using this technique with my method entirely and excessively increases particle pressures leading to stability problems or surface artifacts (Ihmsen et al., 2014a).

2.4.5 Solid Boundary Handling

To handle solid objects including isolated ones without treating solid particles as Poisson particles in the PPE, I aim to compute the Laplacian without including the term $p_i - p_s$, ensuring the solvability of the linear system even with Neumann boundary condition $\frac{dp_i}{d\mathbf{n}_i} = 0$, i.e., $p_i = p_s$ applied. Inspired by IISPH, I decompose the Laplacian operator for solid particles into divergence and gradient as

$$\nabla^2 p_i^{\text{solid}} \approx \nabla \cdot \nabla p_i^{\text{solid}} = -\sum_s V_{is} \frac{\nabla p_i + \nabla p_s}{2} \cdot \nabla W_{is}. \tag{2.7}$$

Assuming $\nabla p_i = \nabla p_s$ when $p_i = p_s$ (Neumann boundary condition) within a local domain because of the smoothed distribution of physical quantities in the SPH setting (Solenthaler and Pajarola, 2009), I obtain

$$\sum_s V_{is} \frac{\nabla p_i + \nabla p_s}{2} \cdot \nabla W_{is} = \nabla p_i \cdot \sum_s V_{is} \nabla W_{is}. \tag{2.8}$$

By using the SPH formulation, I can compute ∇p_i by

$$\nabla p_i = \sum_j V_{ij} \frac{p_i + p_j}{2} \nabla W_{ij} + \sum_s V_{is} \frac{p_i + p_s}{2} \nabla W_{is}.$$

Again, because of the smoothed pressures of fluid particles within their local domain (Solenthaler and Pajarola, 2009), I assume $p_i = p_j$. Combining this assumption and Neumann boundary condition, I obtain

$$\nabla p_i = p_i \left(\sum_j V_{ij} \nabla W_{ij} + \sum_s V_{is} \nabla W_{is} \right). \tag{2.9}$$

I did not gain any improvement in the robustness even if the PPE is solved with $p_i \neq p_j$, and the matrix construction cost significantly increased because of the coefficient of p_j. With Eqs. (2.7), (2.8), and (2.9), I obtain

$$\nabla^2 p_i^{\text{solid}} = b_i p_i,$$
$$b_i = -\left(\sum_j V_{ij}\nabla W_{ij} + \sum_s V_{is}\nabla W_{is}\right) \cdot \sum_s V_{is}\nabla W_{is}. \tag{2.10}$$

Since my solid boundary handling introduces a new term $b_i p_i$ into the PPE when fluid Poisson particles are in contact with Neumann particles, I can change an unsolvable system to a solvable one. For example, when there are two fluid particles i and j touching each other and a solid particle s touching at least one of i and j, the PPE for i and j without my solid handling can be written as $a_{ij}(p_i - p_j) = c_i$ and $a_{ji}(p_j - p_i) = c_j$, respectively, where $a_{ij} = a_{ji} \neq 0$. Therefore, $c_i + c_j = 0$ must hold to ensure the solvability of the PPE for the compatibility condition (Bridson, 2008). In general, however, $c_i + c_j \neq 0$ with the density invariance source term, and thus the linear system is unsolvable. On the other hand, with my solid boundary handling, I can write the PPE as $a_{ij}(p_i - p_j) + b_i p_i = c_i$ and $a_{ji}(p_j - p_i) + b_j p_j = c_j$, where at least one of $b_i \neq 0$ and $b_j \neq 0$ holds, and therefore this system is solvable ($p_i = \frac{a_{ij}(c_i+c_j)+b_j c_i}{a_{ij}(b_i+b_j)+b_i b_j}$ and $p_j = \frac{a_{ij}(c_i+c_j)+b_i c_j}{a_{ij}(b_i+b_j)+b_i b_j}$).

It is worth noting that b_i is generally negative when particle i is compressed toward solid particles, and thus my solid boundary handling is likely to lead to smaller pressures near solid objects, as p_i without my solid handling can be computed by $p_i = \frac{c_i + \sum_j a_{ij} p_j}{\sum_j a_{ij}}$, and $p_i = \frac{c_i + \sum_j a_{ij} p_j}{\sum_j a_{ij} + b_i}$ with my solid handling, where a_{ij} is also negative because of the kernel definition (Müller et al., 2003). However, the blended density can increase particle pressures near solid objects because of the spiky kernel, and effectively prevent particle penetrations.

My solid boundary handling is decoupled from pressures of neighbor fluid and solid particles. This decoupling allows us to significantly simplify my implementation and efficiently compute contributions from solid particles without multiple access to fluid and solid particles and matrix structures specialized for sparse linear systems. In addition, my method does not increase the size of the linear system nor CG iterations, unlike the solid handling used in (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003).

My solid boundary handling is numerically equivalent to implicitly adding Dirichlet boundary condition to the system, to make the system solvable. Thus, in this perspective, my method and the ghost-particle-based

method (Nair and Tomar, 2014) share the same idea. However, my method cannot handle isolated fluid Poisson particles with no channels to Dirichlet particles (this case is handled by setting Dirichlet boundary condition to such particles with my free surface handling), and needs the fluid Poisson particles to be touching Neumann particles to ensure the solvability. On the other hand, the ghost-particle-based method can handle fluid Poisson particles without Neumann and Dirichlet particles at the expense of difficult parameter tuning and underestimated pressures.

2.4.6 PPE Solve

Algorithm 2 summarizes an algorithm for solving the PPE. Since my free surface and solid boundary handling can be incorporated into the originally existing particle loops, the additional cost is trivial.

Algorithm 2 PPE solve

```
for all fluid particle i do
    compute source term c_i
    determine Dirichlet particles with Eq. (2.6)
    if i is Dirichlet particle then
        p_i = 0
    else
        for all fluid particle j do
            compute a_ij for the matrix
            compute V_ij ∇W_ij
        for all solid particle s do
            compute V_is ∇W_is
        compute b_i with Eq. (2.10) for the matrix
solve the linear system using a CG solver
for all fluid particle i do
    if p_i < 0 then
        p_i = 0
```

2.5 Results

I implemented my algorithm with artificial viscosity (Monaghan, 1992), and used incomplete Cholesky CG for solving the linear system. In my method, I use $h = 2d$ (d: initial particle spacing). Light blue and green particles represent fluid and solid particles, respectively. Cyan, magenta, and yellow particles represent Poisson, Dirichlet, and Neumann particles, respectively. I measured computational time for a PC with 4-core 3.40 GHz CPU and RAM 16.0 GB, excluding the surface reconstruction and rendering from the measurement. N^{f} and N^{s} denote the number of fluid and solid particles, respectively. N^{P}, N^{D}, and N^{N}

denote the averaged number of Poisson, Dirichlet, and Neumann particles, respectively. l^{avg} denotes the averaged number of CG iterations per simulation step. t^{p} and t^{f} denote averaged computational time for solving the PPE and for the frame, respectively. Particles are rendered at 60 Hz, and the result of the last simulation step in the frame is used for the color code of particle rendering.

2.5.1 Convergence Criterion

While the residual is commonly used in the Eulerian fluid simulation, a density-based criterion, based on the positive density error $(\rho_i^{\text{err}})^l = \max(0, \rho_i^l - \rho_0)$ (ρ_i^l: estimated density after l iterations), which represents the level of fluid compression, is traditionally used as a convergence criteria in the SPH fluid simulations (Solenthaler and Pajarola, 2009; Ihmsen et al., 2014a). Ihmsen et al. (Ihmsen et al., 2014a) proposed using the average density error, not the maximum one, to keep fluid volumes constant. For the intuitiveness and fair comparison with previous methods, I use the density-based criterion with the averaged positive density error.

In each CG iteration, I can obtain the residual r_i^l without explicitly computing it, following its definition given as $r_i^l = \frac{\rho_0 - (\rho_i^*)^l}{\Delta t^2} - \nabla^2 p_i^l$. Since the estimated density ρ_i^l can be computed by $\rho_i^l = (\rho_i^*)^l + \Delta t^2 \nabla^2 p_i^l$ (Ihmsen et al., 2014a), I can obtain the positive density error with the residual r_i^l as $(\rho_i^{\text{err}})^l = \max(0, \rho_i^l - \rho_0) = \max(0, -r_i^l \Delta t^2)$. Then, I compute the averaged, positive density error $(\hat{\rho}^{\text{err}})^l = \frac{1}{N} \sum_{i \in \Omega^P} (\rho_i^{\text{err}})^l$ (N: the number of Poisson particles). If $(\hat{\rho}^{\text{err}})^l / \rho_0 < \eta$ (η: error threshold), I terminate the CG iterations. I use $\eta = 0.01\%$ in all the scenarios.

2.5.2 Density Blending

I tested my density blending scheme by comparing my method with the density blending and without it (i.e., my method using a smooth or spiky kernel only). Particles are color-coded based on ζ_i (when ρ_i (n_i) computed with the smooth (spiky) kernel is dominant, particle colors approach white (blue)).

Comparison with the smooth kernel. Figure 2.4 compares my method using the density blending and a smooth kernel only. When the time step is large, my method with the smooth kernel suffers from particle penetrations, as shown in Figure 2.4 (b). My method with density blending can prevent penetrations because of the rapidly increased pressure by the spiky kernel, as shown in Figure 2.4 (c), taking 1.36x larger time steps.

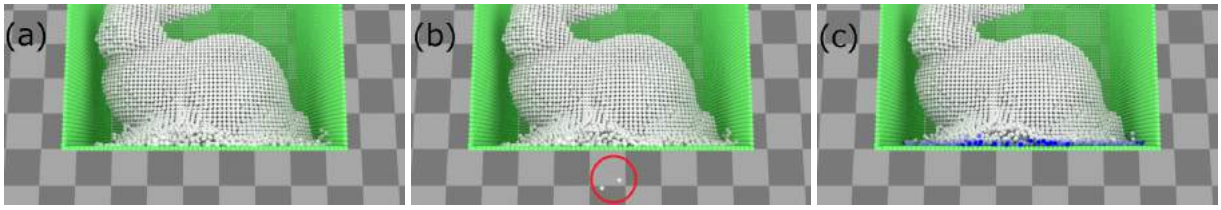


Figure 2.4: Stability test with a bunny drop, where averaged particle spacing is 1.50×10^{-2} m. Smooth kernel with (a) $\Delta t = 1.03 \times 10^{-3}$ s and (b) $\Delta t = 1.40 \times 10^{-3}$ s, where particle penetrations occur as noted by a red circle. (c) Blended density with $\Delta t = 1.40 \times 10^{-3}$ s.

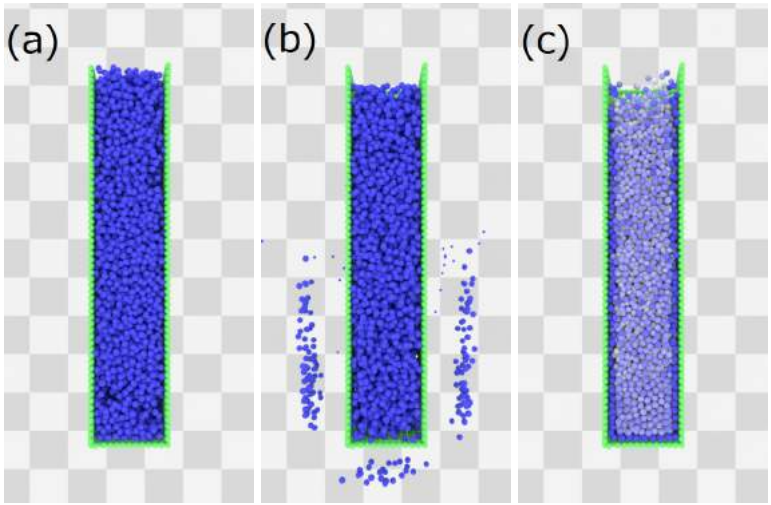


Figure 2.5: Stability test with a fluid column, where averaged particle spacing is 1.50×10^{-2} m. Spiky kernel with (a) $\Delta t = 0.73 \times 10^{-3}$ s and (b) $\Delta t = 1.07 \times 10^{-3}$ s, where particle penetrations occur. (c) Blended density with $\Delta t = 1.07 \times 10^{-3}$ s.

Comparison with the spiky kernel. Figure 2.5 compares my method using the density blending and a spiky kernel only, where very large time steps are chosen to clearly show the differences in the robustness. When the time step is large, my method with the spiky kernel can become unstable and causes particle penetrations, as shown in Figure 2.5 (b). My method with density blending does not cause particle penetrations because of the blended smooth kernel, as shown in Figure 2.5 (c), taking 1.46x larger time steps.

2.5.3 Free Surface Handling

I tested my free surface handling to demonstrate its robustness with a simple scene, where a cubed fluid is dropped from a lower position to make the free surface handling crucial in determining time steps (see Figure 2.6 (a) for the initial setup). I compared my method with the density-based (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003), surface-based (He et al., 2012b), ghost-particle-based (Nair and Tomar, 2014), and source-term-based method, which treats fluid particles as Dirichlet particles if $\rho_0 - \rho_i^* \geq 0$,

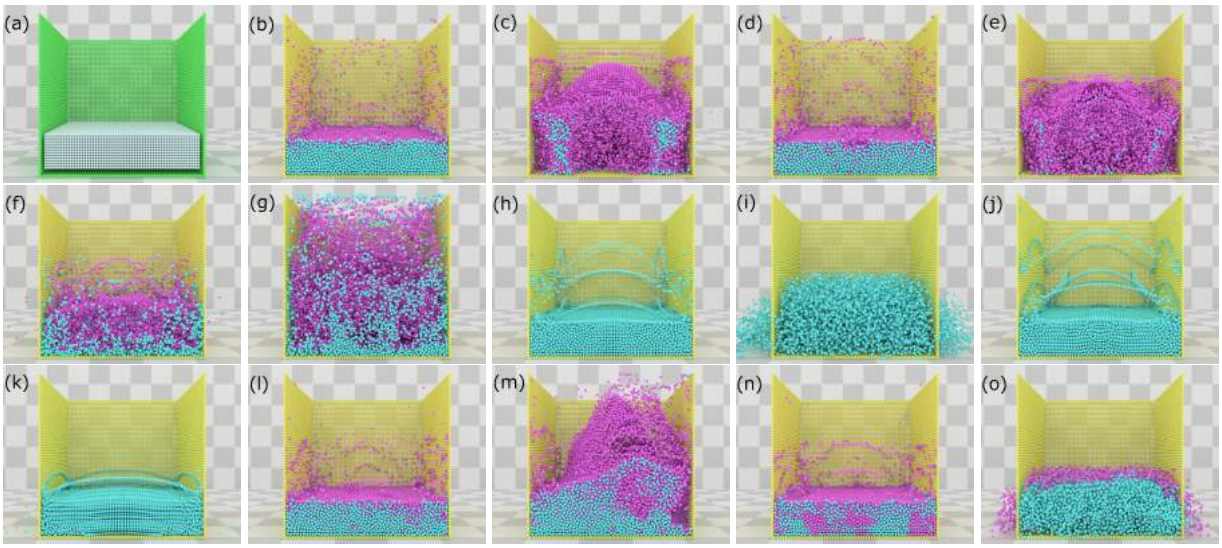


Figure 2.6: Comparison for free surface handling, where averaged particle spacing is 1.50×10^{-2} m. Different frames are chosen for illustration. (a) Initial setup. Density-based with (b) $\Delta t = 0.75 \times 10^{-3}$ s (stable) and (c) $\Delta t = 1.68 \times 10^{-3}$ s (unstable). Density-based (clamped) with (d) $\Delta t = 0.75 \times 10^{-3}$ s (stable) and (e) $\Delta t = 1.68 \times 10^{-3}$ s (unstable). (f) Surface-based with $\Delta t = 0.35 \times 10^{-3}$ s (unstable). (g) Surface-based (clamped) with $\Delta t = 0.35 \times 10^{-3}$ s (unstable). Ghost-particle-based with (h) $\Delta t = 0.42 \times 10^{-3}$ s (stable) and (i) $\Delta t = 1.68 \times 10^{-3}$ s (unstable). Ghost-particle-based (clamped) with (j) $\Delta t = 0.52 \times 10^{-3}$ s (stable) and (k) $\Delta t = 1.68 \times 10^{-3}$ s (unstable). Source-term-based with (l) $\Delta t = 1.20 \times 10^{-3}$ s (stable) and (m) $\Delta t = 1.68 \times 10^{-3}$ s (unstable). (n) Mine with $\Delta t = 1.68 \times 10^{-3}$ s (stable). (o) Mine (clamped) with $\Delta t = 1.68 \times 10^{-3}$ s (unstable).

combining the clamping approach (Ihmsen et al., 2014a). For the comparisons, I used my solid boundary handling. Figure 2.6 illustrates the results of my method and others.

Density-based method. While the density-based method with a small time step as in Figure 2.6 (b) generates plausible fluid behaviors, the method suffered from a stability problem with a large time step seen in Figure 2.6 (c). The clamping did not improve nor deteriorate the robustness of this method (see Figure 2.6 (d) and (e)).

Surface-based method. The surface-based method sets Dirichlet particles only near surfaces, and consequently particle pressures are likely to be excessively high inside of the fluid volume. Thus, this method failed to perform a stable simulation even with a small time step as shown in Figure 2.6 (f). Since the clamping approach increases particle pressures, this technique did not improve the stability as shown in Figure 2.6 (g).

Ghost-particle-based method. Because of the underestimated pressures, a small time step was necessary to ensure no particle penetrations as in Figure 2.6 (h). This method failed with a large time step as in Figure 2.6

Table 2.4: Performance comparison for Figure 2.6. My method achieves the best performance, taking the largest time step due to the improved robustness.

Figure	Method	N^{f}	N^{s}	N^{P}	N^{D}	N^{N}	$\Delta t(s)$	l^{avg}	t^{p} (s)	t^{f} (s)
2.6 (b)	Density-based	42.2k	21.8k	40.0k	2.3k	21.8k	0.75×10^{-3}	2.92	4.79	6.64
2.6 (j)	Ghost-particle-based (clamped)	42.2k	21.8k	42.2k	0.0k	21.8k	0.52×10^{-3}	0.99	5.95	8.85
2.6 (l)	Source-term-based	42.2k	21.8k	32.0k	10.3k	21.8k	1.20×10^{-3}	1.96	2.20	3.23
2.6 (n)	Mine	42.2k	21.8k	28.3k	14.0k	21.8k	1.68×10^{-3}	4.66	2.16	2.97

(i). The increased pressure using the clamping slightly improved the robustness of this method, allowing for the use of a larger time step as seen in Figure 2.6 (j). The clamping was insufficient to take a very larger time step as in Figure 2.6 (k). For all the scenes, I observed the volume change (fluid oscillation) because of underestimated pressures even though the convergence criterion is satisfied.

Source-term-based method. The source-term-based method can perform a stable simulation with a moderately large time step as in Figure 2.6 (l) while the simulation becomes unstable with a large time step as in Figure 2.6 (m). The clamping approach is ineffective for this method because Dirichlet particles determined by the source term are excluded from the system.

My method. My method can perform a stable simulation with a large time step as in Figure 2.6 (n), while the clamping introduced stability issues due to increased pressures (see Figure 2.6 (o)).

Table 2.4 shows simulation conditions and performance for Figures 2.6 (b), (j), (l), and (n), where the density-based, ghost-particle-based, source-term-based, and my method generate plausible fluid behaviors with their best performance, respectively. Since my method can take larger time steps than the others, my method outperformed the others regardless of the more iterations required for convergence, amortizing the increased cost of the pressure solve. When I need additional computations, e.g., for viscosity, stress, temperature, and surface tension, taking larger time steps can further increase the performance gain of my method.

The number of Poisson (Dirichlet) particles with my method is smaller (larger) than that of the source-term-based method in Table 2.4. However, this is due to different time steps, and the number of Poisson and Dirichlet particles was 34.4k ($>$ 32.0k) and 7.8k ($<$ 10.3k), respectively, when the same time step as in Figure 2.6 (l) was used with my method.

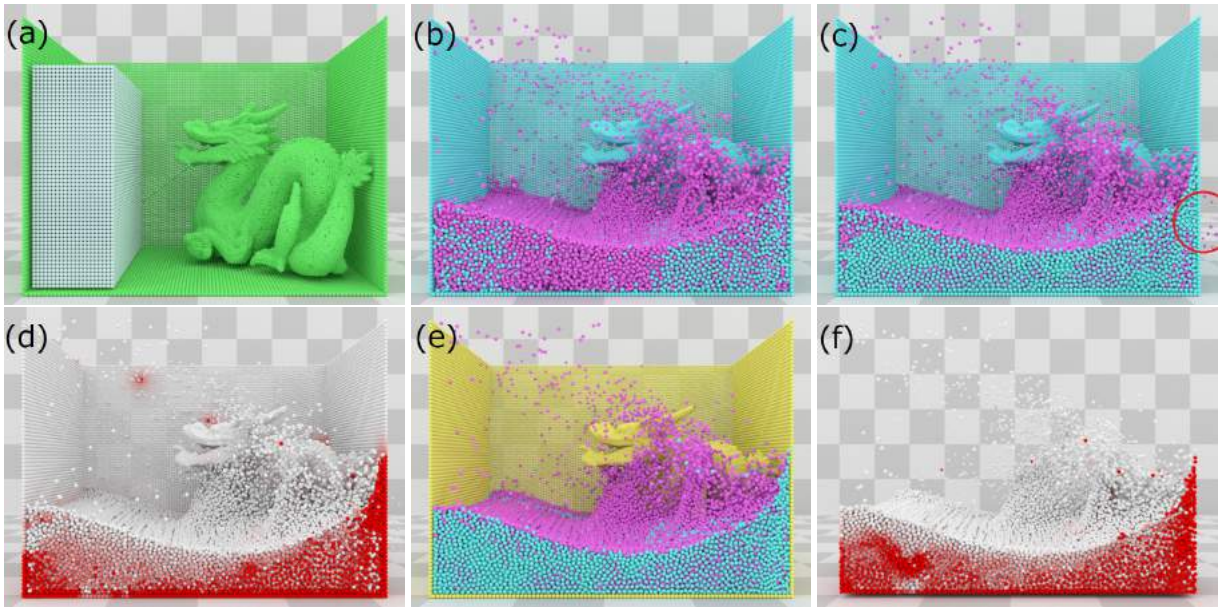


Figure 2.7: Comparison for solid boundary handling, where averaged particle spacing is 1.29×10^{-2} m. (a) Initial setup. Previous method with (b) $\Delta t = 0.51 \times 10^{-3}$ s and (c) with $\Delta t = 1.09 \times 10^{-3}$ s, where particle penetrations occur as noted by a red circle; and (d) with particles color coded based on their pressures. (e) My method with $\Delta t = 1.09 \times 10^{-3}$ s, and (f) with particles color coded based on their pressures. Red and white particles represent high and low pressures, respectively.

2.5.4 Solid Boundary Handling

I tested my solid handling method to demonstrate its robustness and efficiency with a dam break scene with a solid dragon (see Figure 2.7 (a) for the initial setup). Note that the dragon must be connected to the surrounding cube representing the simulation domain to ensure the solvability of the linear system for (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003). I compare these methods in Figure 2.7, adopting my free surface handling for both methods, and show simulation conditions and performance in Table 2.5.

While both methods can generate plausible fluid behaviors, the increased number of Poisson particles with the previous method leads to a larger size of the linear system, requiring more computations. Additionally, the previous method erroneously underestimates particle pressures, and the averaged maximum pressure was 1.88×10^4 and 2.32×10^4 $\mathrm{kg}/(\mathrm{m} \cdot \mathrm{s}^2)$ in Figures 2.7 (d) and (f), respectively, with the same time step $\Delta t = 1.09 \times 10^{-3}$ s. Thus, the previous method (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) needed to use smaller time steps than those for my method to prevent particle penetrations. Consequently, my method outperformed the previous method by a factor of 2.20.

Table 2.5: Performance comparison for Figure 2.7. My method outperforms the previous method by taking a 2.14x larger time step, and achieves the performance gain by a factor of 2.20.

Figure	Method	N^{f}	N^{s}	N^{P}	N^{D}	N^{N}	$\Delta t(s)$	l^{avg}	t^{p} (s)	t^{f} (s)
2.7 (b)	Previous method	76.6k	57.1k	90.6k	43.1k	0.0k	0.51×10^{-3}	2.26	16.62	26.37
2.7 (e)	My method	76.6k	57.1k	59.1k	17.5k	57.1k	1.09×10^{-3}	4.39	6.99	11.97

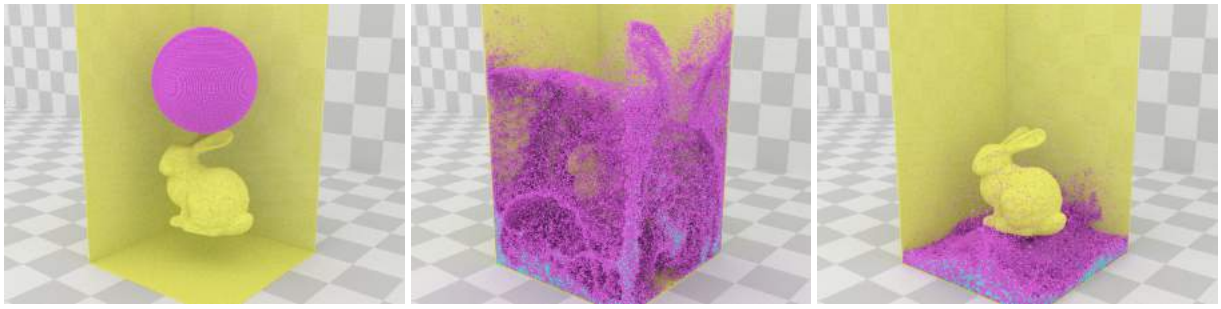


Figure 2.8: A fluid drop with a floating solid bunny.

Additionally, my method can handle solid objects floating in the air (see Figure 2.8).

2.5.5 Comparison with IISPH

I compare my method with IISPH (Ihmsen et al., 2014a) using a dam break scenario, as shown in Figure 2.9 for the fluid behaviors and Figure 2.10 for their performance of the pressure solve. While both methods generate comparable results, my method outperformed by a factor of 3.78 in the pressure solve (IISPH used 86.73 s while mine used 22.95 s) because of the fast convergence of CG.

2.5.6 Time Step Scaling

To compare the performance with different time steps, I experimented with a dam break scene, shown in Figure 2.9. For this comparison, averaged particle spacing was 1.80×10^{-2} m. Table 2.6 shows performance results with different time steps. With my method, the number of iterations increases *sublinearly* according to time steps. The sublinearity offers an advantage that my method generally achieves the best performance with the largest available time steps, simplifying a process of finding optimal time steps. By contrast, the number of iterations for IISPH with weighted Jacobi increases superlinearly (Ihmsen et al., 2014a), and thus IISPH needs to take into account various factors, e.g., pressure solve and neighbor search, to determine time steps for the optimal performance of IISPH.

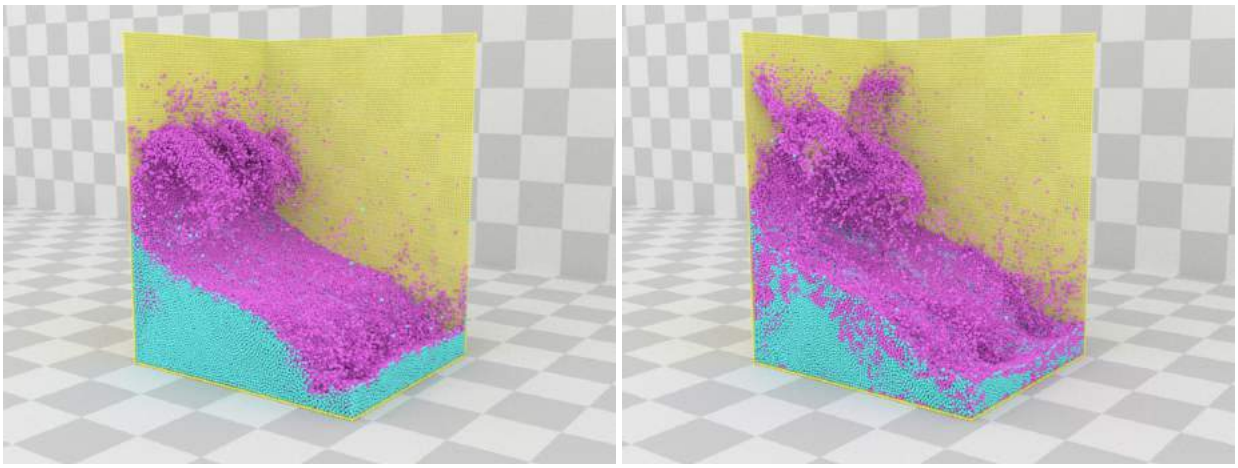


Figure 2.9: Comparison of fluid behaviors. (Left) IISPH. (Right) My method.

Table 2.6: Performance results with different time steps. With my method, l^{avg} increases sublinearly w.r.t. time steps.

$\Delta t(s)$	l^{avg}	$t^{\text{p}}(s)$	$t^{\text{f}}(s)$
0.20×10^{-3}	0.99	12.27	24.93
1.60×10^{-3}	6.69	2.96	4.69

2.6 Discussions and Limitations

In this section, I discuss applicability of CG to IISPH (§ 2.6.1), and limitations of my solver (§ 2.6.2).

2.6.1 CG for IISPH

In (Ihmsen et al., 2014a), the authors attempted to solve the PPE using CG with the assumption of uniformly constant particle mass and density to make the system symmetric. With the IISPH discretization, however, the system can be not diagonally dominant and thus not positive definite.

The left hand side of the PPE in the IISPH can be written as

$$-\frac{m^2}{\rho_0^2}\sum_j\left(\sum_j(p_i+p_j)\nabla W_{ij}-\sum_k(p_j+p_k)\nabla W_{jk}\right)\nabla W_{ij} \tag{2.11}$$

(j: first-ring neighbors and k: second-ring neighbors), including second-ring neighbors. According to (Takahashi et al., 2015), I can separately write terms on p_i, p_j, and p_k with $\omega_{ij} = \sum_j \nabla W_{ij}$ as $-\frac{m^2}{\rho_0^2}\|\omega_{ij}\|^2 p_i$, $-\frac{m^2}{\rho_0^2}\sum_j \nabla W_{ij}(\omega_{ij}-\omega_{jk})p_j$, and $-\frac{m^2}{\rho_0^2}\omega_{ij}\sum_k \nabla W_{jk}p_k$, respectively, and the diagonal component is the coefficient of p_i, which is $-\frac{m^2}{\rho_0^2}\omega_{ij}\cdot(\omega_{ij}+\sum_j \nabla W_{ji})$ taking contributions of k into account (particle k can

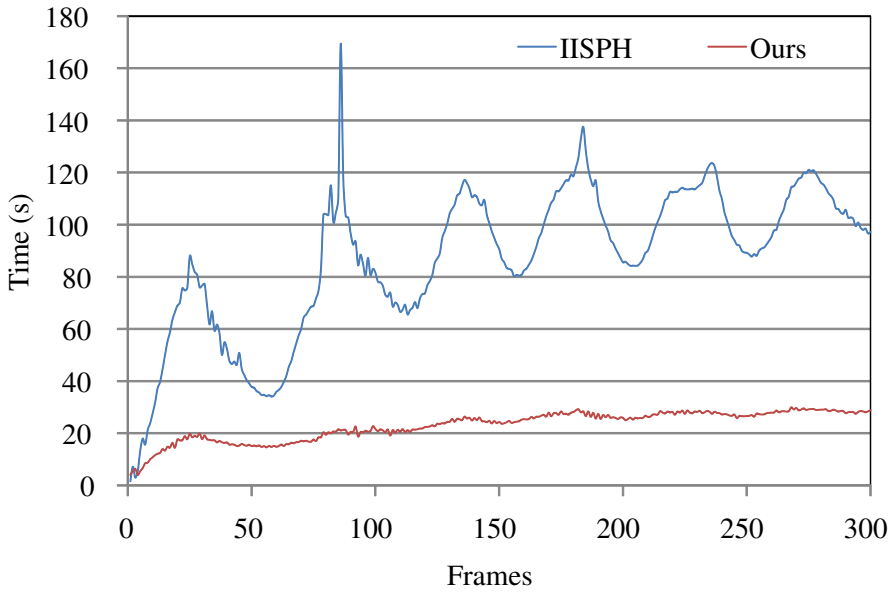


Figure 2.10: Comparison of pressure solve time for Figure 2.9. My method outperforms IISPH by a factor of 3.78.

be particle i). When particle i has a completely uniform neighbor particles ($\omega_{ij} = 0$) and j does not have ($\omega_{jk} \neq 0$), this system is not diagonally dominant. Therefore, this system cannot be solved with CG.

It is worth noting that Jacobi method also cannot solve a non-diagonally-dominant system, and thus *weighted* Jacobi method is used in (Ihmsen et al., 2014a).

2.6.2 Limitations

My density blending and interface handling methods significantly improve the stability, and the improved robustness of my solver can be comparable to IISPH depending on scenarios. However, my solver can be less robust than IISPH when very large time steps and high resolutions are used, since IISPH considers pressure forces with farther particles in the PPE, generating more reliable pressures. Although there are many situations where my method is more advantageous (e.g., when I cannot use very large time steps because of fast moving particles), the weaker robustness is a limitation and should be investigated to make my solver more robust. Additionally, IISPH can be advantageous for relatively smaller scenarios consisting of shallow water only, with a soft constraint on volume changes, because fewer Jacobi iterations can be sufficient to converge, and thus my method cannot benefit from CG, which shows a fast, superlinear convergence in the latter phase.

my density blending uses two tunable parameters. Although I was able to use constant values (as suggested in Section 2.3) in my scenarios, when the distributions of solid particles are highly irregular, it would be necessary to take into account actual contributions from solid particles. Other than blending, it might also be possible to design a new shape of kernels to address this issue.

2.7 Conclusions and Future Work

I proposed a hybrid incompressible SPH solver with a new interface handling method. My density blending improves the stability for both fluid-fluid and fluid-solid collisions. My free surface handling improves the robustness by appropriately setting Dirichlet particles with Jacobi-based pressure prediction while my solid boundary handling introduces a new term to ensure the solvability of the linear system even with objects floating in the air, without sacrificing memory and computational efficiency. Several examples demonstrated the effectiveness of my method.

In general, my solver becomes more efficient as I take larger time steps unlike IISPH with weighted Jacobi since required iterations sublinearly increase according to time steps. Adopting a more effective preconditioner would further accelerate my method, solving the linear system with fewer iterations even if larger time steps make the system ill-conditioned. Additionally, such a preconditioner might allow us to handle larger scale scenarios with deep water depth, for which both of my solver with CG and IISPH with weighted Jacobi show a poor scalability.

It is interesting to adopt different approaches to ensure the solvability of the linear system. I plan to use ghost SPH (Schechter and Bridson, 2012) to generate particles outside of fluid volumes, which can be used for Dirichlet boundary condition. Though this is similar to (Nair and Tomar, 2014), I can use better sampled particles with the ghost SPH, and this would further improve the robustness. To simulate fluids with my method in a closed container, i.e., Neumann boundary condition only (without free surfaces, i.e., Dirichlet boundary condition), adopting the source term correction approach described in (Bridson, 2008) would be promising.

Acknowledgements

This work is supported in part by JASSO for Study Abroad, JST CREST, U.S. National Science Foundation, and UNC Arts and Sciences Foundation. I would like to thank Matthias Teschner and anonymous

reviewers for their valuable suggestions and comments. I also thank Pavel Krajcevski for helping to proofread the final version of this work.

CHAPTER 3: A Multilevel SPH Solver with Unified Solid Boundary Handling

3.1 Introduction

Lagrangian particle-based methods, such as Smoothed Particle Hydrodynamics (SPH), have become popular because of the attractive advantages of particles, e.g., conceptually simpler discretization and collision handling at the particle level, and thus particle-based methods have been extensively adopted in fluid simulation (Ihmsen et al., 2014b). To generate realistic liquid behaviors, enforcing liquid incompressibility is essential, and various particle-based methods have been proposed (Koshizuka et al., 1996; Cummins and Rudman, 1999; Becker and Teschner, 2007b; Sin et al., 2009; Solenthaler and Pajarola, 2009; Raveendran et al., 2011; He et al., 2012a; Bodin et al., 2012; Macklin and Müller, 2013; Ihmsen et al., 2014a; Bender and Koschier, 2015). In particular, solving a pressure Poisson equation (PPE) has been proven to be an effective approach, and the PPE is commonly solved with a stationary iterative solver (e.g., Jacobi method) or Krylov method (e.g., Conjugate Gradient (CG)). In addition to enforcing the incompressibility, discretizing fluids at a high resolution is also important and contributes to the quality of liquid behaviors avoiding numerical dissipations and producing fine details near fluid surfaces. However, enforcing the incompressibility under high-resolution discretization is likely to be computationally challenging because previously used solvers (e.g., Jacobi method and CG) do not scale well and require more iterations to solve the PPE as the number of particles increases.

Figure 3.1: High-resolution incompressible fluids simulated with my multilevel solver. My method outperforms other particle-based solvers in the pressure solve, and its computational cost scales nearly linearly with respect to the number of particles.

To address this problem, in the Eulerian grid-based approach, scalable multigrid (MG) methods have been used (McAdams et al., 2010; Chentanez and Müller, 2011; Chentanez and Müller, 2012; Ferstl et al., 2014; Weber et al., 2015) with the regular Cartesian grid, which can also be used to construct the hierarchy. This multilevel approach is also adopted for deformable body simulation by constructing the hierarchy from embedded grid structures (Zhu et al., 2010; McAdams et al., 2011) or computing nearly optimal coarser-level structures from finer-levels (Müller, 2008; Wang et al., 2010). Unlike deformable objects, however, it is difficult to apply the multilevel approach to particle-based methods because the connectivity of particles changes at every simulation step, requiring expensive remeshing to keep the quality of coarser-level meshes and to avoid mesh tangling. Although Cummins and Rudman (Cummins and Rudman, 1999) adopted a fixed Cartesian grid for particle-based fluid simulation, solutions at the particle and grid levels are generally inconsistent because of the different discretization approaches, and thus their method can diverge, stagnate, or fail to achieve optimal efficiency of multilevel solvers.

I propose a new multilevel method for efficiently solving the PPE arising from the particle-based fluid simulation. My method offers the following technical contributions:

- **A multilevel method for particle systems** that constructs hierarchies, establishes the correspondence between solutions at the particle and grid levels, and coarsens simulation elements taking boundary conditions into account.
- **A new solid-handling method for the PPE** to ensure the solvability of the system regardless of the particle configurations while approximating the original solution to make it solvable with my multilevel approach.

My solver is able to achieve up to one order of magnitude performance gain in the pressure solve as compared to one of the state-of-the-art particle-based solver, implicit incompressible SPH (IISPH) (Ihmsen et al., 2014a), and the cost of my method scales nearly linearly with respect to the number of unknowns in the system, considerably outperforming existing particle-based methods.

3.2 Related Work

Particle-based methods have been extensively studied, and I refer to (Ihmsen et al., 2014b) for their basis and applications. I focus my discussions on particle-based methods closely related to mine.

3.2.1 Incompressibility

Early works used an equation of state to compute pressures from densities (Desbrun and Gascuel, 1996; Müller et al., 2003). In particular, SPH methods using the Tait equation (Monaghan, 1994; Becker and Teschner, 2007b), known as weakly compressible SPH (WCSPH), can generate higher pressures mitigating the volume compression of fluids, and thus have been widely adopted even in recent SPH works because of the simplicity and effectiveness (He et al., 2014; Ren et al., 2014). However, since WCSPH usually produces excessively high pressures and thus strong pressure forces, smaller time steps are necessary for stable fluid simulations. Additionally, it is also undesirable that WCSPH cannot explicitly specify the tolerance of the fluid compression.

To address these issues, several approaches that globally solve a system with iterative methods have been developed. Adopting the pressure projection commonly used in the Eulerian approach (Bridson, 2015), incompressible SPH (ISPH) (Cummins and Rudman, 1999; Shao and Lo, 2003) and Moving Particle Semi-implicit (MPS) (Koshizuka et al., 1996; Premoze et al., 2003) were proposed. Similar projection-based approaches were also proposed using a Voronoi diagram (Sin et al., 2009) and a power diagram (de Goes et al., 2015).

Another recent trend is to locally solve a system with iterative methods. Predictive-corrective incompressible SPH (PCISPH), which iteratively predicts and corrects particle density in a Jacobi manner, was proposed by Solenthaler and Pajarola (Solenthaler and Pajarola, 2009). This predictive-corrective scheme was also adopted in local Poisson SPH (He et al., 2012a). Macklin and Müller (Macklin and Müller, 2013) proposed position-based fluids (PBF) adopting position-based dynamics (PBD) (Müller et al., 2007) for density constraints, and improved the robustness. Another constraint-based solver was also proposed by Bodin et al. (Bodin et al., 2012) to improve the accuracy. To further improve the efficiency and robustness compared to PCISPH, Ihmsen et al. (Ihmsen et al., 2014a) proposed IISPH, which decomposes the Laplacian operator in the PPE into the divergence and gradient operators to accelerate the propagation of updated pressures with the Jacobi method. However, since IISPH generates not diagonally dominant systems, it does not produce smooth pressure fields, and CG (which is generally faster than the Jacobi method) cannot be applied to IISPH (Takahashi et al., 2016). In (Ihmsen et al., 2014a), it is demonstrated that IISPH outperforms ISPH when both approaches use the Jacobi method. Cornelis et al. (Cornelis et al., 2014) proposed combining IISPH with Fluid Implicit Particle (FLIP) (Zhu and Bridson, 2005) to further improve the performance.

Bender and Koschier (Bender and Koschier, 2015) proposed divergence-free SPH, which enforces not only the density-invariance condition but also divergence-free condition, similar to (Kang and Sagong, 2014), to reduce the density deviations at the next step.

3.2.2 Multilevel Particles

To improve the efficiency by reducing the number of particles, approaches using spatially adaptive particles have been proposed. Adams et al. (Adams et al., 2007) used different sizes of particles to allocate more computational resources to regions, which are important in terms of visual quality and fluid dynamics, e.g., surfaces and near solid objects. This method was extended to improve the robustness by blending particle properties over time (Orthmann and Kolb, 2012). To avoid direct interactions between particles at different levels, Solenthaler and Gross (Solenthaler and Gross, 2011) proposed separating domains for fine- and coarse-scale particles. Although these approaches can reduce the number of particles, energy dissipation (i.e., damping) can be introduced into the simulation because of the coarser level particles and may negatively affect liquid behaviors.

Cummins and Rudman (Cummins and Rudman, 1999) used MG to solve the PPE with ISPH adopting the Cartesian grid for the hierarchy construction. However, they did not include a Dirichlet boundary condition at all, which is necessary to handle general liquid simulation scenarios. In addition, they used a mirroring approach for the Neumann boundary condition that copies physical quantities to the opposite side of solid boundaries, limiting its applicability to rectangular domains only. Thus, how to address complex Dirichlet and Neumann boundary conditions in the MG setting is unclear. Moreover, they did not address discrepancies of solutions at the particle and grid levels, which can occur due to various factors, such as different discretization methods, particle irregularities, and kernel definitions. Consequently, this approach can diverge, stagnate, or fail to achieve the optimal efficiency of MG. Their experiments were limited to 2D simulations and were never tested on fine-scale scenarios. Thus, how this method works in such scenarios was not demonstrated. The method of Raveendran et al. (Raveendran et al., 2011) can be considered as a two-level approach consisting of the finest particle level and the coarsest grid level, and uses the result of the pressure projection on the grid to accelerate the convergence of an iterative solver on the particles. However, since the coarse pressure projection is not sufficiently accurate and does not consider the solution discrepancies, this approach cannot significantly reduce the number of solver iterations on the particles.

3.3 SPH Fluid Simulation

I first briefly describe the simulation algorithm of my fluid solver, which is similar to that of the Eulerian approach (Section 3.3.1), and then explain the PPE for particle-based fluids clarifying differences from the Eulerian approach (Section 3.3.2).

As my underlying fluid solver, I employ ISPH (Cummins and Rudman, 1999; Shao and Lo, 2003) since ISPH constructs a linear system whose matrix is symmetric positive (semi-) definite (SPD) and produces smooth pressures allowing us to solve the system with either CG or MG. Other methods, which have properties similar to ISPH, can be used as well.

3.3.1 Simulation Algorithm

In the Lagrangian setting, incompressible flow for particles can be described by the continuity equation $\frac{d\rho_i}{dt} + \rho_i \nabla \cdot \mathbf{u}_i = 0$, and the Navier-Stokes equations $\frac{d\mathbf{u}_i}{dt} = -\frac{1}{\rho_i}\nabla p_i + \frac{\mathbf{F}_i^{\mathrm{v}}}{m_i} + \frac{\mathbf{F}_i^{\mathrm{ext}}}{m_i}$, where ρ_i denotes density of particle i, t time, $\mathbf{u}_i$ velocity, p_i pressure, $\mathbf{F}_i^{\mathrm{v}}$ viscosity force, m_i mass, and $\mathbf{F}_i^{\mathrm{ext}}$ external force. Similar to the Eulerian approach, I take the operator splitting to enforce the fluid incompressibility. First, I find neighbor fluid and solid particles to compute the density. Then, I predict the intermediate velocity $\mathbf{u}_i^*$ and the intermediate density ρ_i^*. Next, I solve the PPE to obtain pressure. Finally, I compute pressure forces and integrate velocities and positions. For more algorithm details, I refer to Section 3.4 of the survey paper (Ihmsen et al., 2014b).

3.3.2 PPE for Particle-based Methods

Unlike the traditional Eulerian approach that enforces the divergence-free condition (Bridson, 2015), I use the density-invariance condition to avoid the volume drift, as commonly employed in the particle-based methods (Koshizuka et al., 1996; Shao and Lo, 2003; Ihmsen et al., 2014a; Bender and Koschier, 2015) . By combining the velocity change caused by pressure forces $\frac{d\mathbf{u}_i}{dt} = -\frac{1}{\rho_i^{t+1}}\nabla p_i$ and the density prediction based on the continuity equation $\frac{\rho_i^{t+1} - \rho_i^*}{\Delta t} = -\rho_i^{t+1}\nabla \cdot \mathbf{u}_i^*$, ($\Delta t$: time step), I obtain the PPE as $-\nabla^2 p_i = \frac{\rho_i^* - \rho_0}{\Delta t^2}$ (ρ_0: rest density) with Dirichlet boundary condition $p_i = 0$ on free surfaces and Neumann boundary condition $\frac{dp_i}{d\mathbf{n}_i} = 0$ ($\mathbf{n}_i$: normal) on solid boundaries (Shao and Lo, 2003). This PPE can be rewritten in the matrix form as $\mathbf{Ap} = \mathbf{b}$, where $\mathbf{A}$ denotes a coefficient matrix, and $\mathbf{p}$ and $\mathbf{b}$ concatenation of pressures and source terms (right hand side of the PPE), respectively. Since the PPE uses the density-invariance condition, which

generally does not satisfy the compatibility condition (i.e., the summation of right hand side is not equal to 0), and $\mathbf{A}$ has null-space (i.e., $\mathbf{A}$ is rank deficient) when a standard Laplacian discretization is used (as done in (Cummins and Rudman, 1999; Shao and Lo, 2003; Koshizuka et al., 1996; Premoze et al., 2003)), a Dirichlet boundary condition is necessary for each of the groups consisting of neighboring particles to ensure the solvability of the PPE (Bridson, 2015; Takahashi et al., 2016).

Because of the negative source term, which is likely to occur at particles with a smaller number of neighbor particles (e.g., near free surfaces), solving the PPE without special care generates negative pressures leading to the tensile instability in particle-based methods (Monaghan, 2000; Schechter and Bridson, 2012; He et al., 2014). Therefore, $\mathbf{p} \geq 0$ must be simultaneously enforced to avoid the tensile instability turning the linear system $\mathbf{Ap} = \mathbf{b}$ into a Linear Complementarity Problem (LCP) $\mathbf{Ap} = \mathbf{b} \perp \mathbf{p} \geq 0$. Although stationary iterative methods can relatively easily solve the LCP with the clamping approach (e.g., projected Jacobi), solving the LCP is numerically more difficult than linear systems, especially with Krylov methods, which prohibit clamping negative pressures in each iteration (Ihmsen et al., 2014a), and thus require specialized techniques (Dostal and Schoberl, 2005). Even with MG solvers, convergence can be delayed (Chentanez and Müller, 2012).

Since solving the LCP is more costly and complex, I approximate the solutions of the LCP by solving a linear system with appropriately set a Dirichlet boundary condition, similar to (Koshizuka et al., 1996; Shao and Lo, 2003). To ensure $\mathbf{p} \geq 0$, I treat fluid particles whose source term is negative as Dirichlet boundary condition (see (Takahashi et al., 2016)). Note that this treatment enforces $\mathbf{b} \geq 0$, and thus $\mathbf{p} \geq 0$ is guaranteed. In practice, I set particles as a Dirichlet boundary condition if $\rho_i^* < 0.99\rho_0$ to avoid erroneous classifications because of the particle irregularity and clamp negative pressures to 0 after the pressure solve to avoid the tensile instability and particle adhesion.

3.4 Solid Boundary Handling

I first describe the PPE discretization based on ISPH, its issues, and previous approaches (Section 3.4.1), and then present my solid boundary handling method (Section 3.4.2). Note that my solid boundary handling can be used without MG while it allows us to solve the PPE with MG by approximating the original solution.

To solve the PPE with solid boundaries on moving particles, it is necessary to adaptively assign roles to the particles, and I classify particles as follows. I call solid particles for the Neumann boundary condition

Neumann particles (rendered as beige), fluid particles without any neighbors *isolated particles* (magenta), fluid particles used for the Dirichlet boundary condition $p_i = 0$ (i.e., if $\rho_i^* < 0.99\rho_0$) *Dirichlet particles* (blue), fluid particles without fluid neighbors and with solid neighbors *separated particles* (green), and otherwise *Poisson particles* (cyan). Since isolated particles always satisfy the condition of Dirichlet particles, I set pressures of isolated particles as 0, similar to Dirichlet particles. While I analytically set pressures of separated particles excluding them from the linear system (see Section 3.4.2), I treat pressures of Poisson particles as unknown variables.

3.4.1 ISPH Discretization

According to (Ihmsen et al., 2014b), $-\nabla^2 p_i$ is discretized as $\sum_j a_{ij}(p_i - p_j)$, where $a_{ij} = -(V_i + V_j)\frac{\mathbf{x}_{ij}\cdot\nabla W_{ij}}{\|\mathbf{x}_{ij}\|^2+0.01h^2} > 0$ with the kernel definition in (Müller et al., 2003) (j: index for fluid neighbors, V: volume, $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, and h: kernel radius). When I consider fluid and solid particles assuming that solid particle pressures are definable, $-\nabla^2 p_i = -\nabla^2 p_i^{\text{fluid}} - \nabla^2 p_i^{\text{solid}} = \sum_j a_{ij}(p_i - p_j) + \sum_s a_{is}(p_i - p_s)$ (s: index for solid neighbors). When Neumann boundary condition $\frac{dp_i}{d\mathbf{n}_i} = 0$, i.e., $p_i = p_s$, is applied, I obtain $-\nabla^2 p_i = \sum_j a_{ij}(p_i - p_j)$.

With this formulation, unfortunately, particle configurations that cannot determine p_i occur because the PPE does not satisfy the compatibility condition. Specifically, p_i cannot be determined, when particle i is a separated particle, or a Poisson particle with no channel (via other neighboring Poisson particles) to at least one Dirichlet particle (Takahashi et al., 2016). One possible approach is to separately address these particles after I check particle connectivities. However, it is hard to efficiently and exactly check particle connectivities (e.g., using the flood fill) because we need to propagate geometric information one by one over all particles. If particle pressures cannot be determined, for example, a widely used collision handling method proposed in (Akinci et al., 2012), which depends on fluid particle pressures, cannot be used.

Some previous works (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) converted Neumann particles to Poisson particles to make the PPE solvable by connecting Poisson and separated particles via converted Poisson particles. However, this approach has some limitations (Takahashi et al., 2016). First, since Neumann particles are incorporated into the linear system, the size of the system becomes larger leading to increased memory and computation cost. Second, due to the solid particles newly incorporated in the system, pressures can be underestimated resulting in particle penetrations with (Akinci et al., 2012). Third, this approach cannot solve a linear system with solid objects floating in the air, which

can frequently occur in two-way interactions of fluids and solids, because converted Poisson particles in the objects may not have a channel to Dirichlet particles.

To address these issues, Takahashi et al. (Takahashi et al., 2016) proposed introducing a new pressure term, which generally increases the diagonal components, to make the system solvable. However, their approach does not ensure the diagonal dominance of the resulting linear system, and thus MG and CG may fail to converge. Additionally, the increased diagonal components may erroneously underestimate pressures.

3.4.2 Unified Handling for Solid Boundary

To handle various simulation scenarios, I take an approach that can always ensure the solvability of the linear system in a unified manner regardless of particle configurations. My method introduces a new term, which increases the diagonal component, into the left hand side of the PPE to make it solvable while amplifying the right hand side to counteract underestimated pressures compared to the solutions, which can be obtained from the original linear system.

For my solid boundary handling scheme, I employ the method of (Akinci et al., 2012). This method puts one layer of particles on solid boundaries defining $\delta_i = \frac{1}{\sum_s W_{is}}$, which adjusts the contribution of solid particles based on their sampling density to alleviate over sampling (see (Akinci et al., 2012) for details). According to (Ihmsen et al., 2014a), when the incompressibility is enforced by the pressure force from solid particles $\mathbf{F}_i^{\mathrm{p,solid}}$, the continuity equation can be discretized as

$$\frac{\rho_0 - \rho_i^*}{\Delta t} = \sum_s \frac{\rho_0}{\delta_s} \Delta \mathbf{u}_i \cdot \nabla W_{is} = \rho_0 \Delta \mathbf{u}_i \cdot \sum_s \frac{1}{\delta_s} \nabla W_{is}, \tag{3.1}$$

where $\Delta \mathbf{u}_i$ denotes the velocity change due to $\mathbf{F}_i^{\mathrm{p,solid}}$ (i.e., $\Delta \mathbf{u}_i = \frac{\mathbf{F}_i^{\mathrm{p,solid}}}{m_i} \Delta t$). In (Akinci et al., 2012), the pressure force from solid particles is defined as $\mathbf{F}_i^{\mathrm{p,solid}} = -m_i \sum_s \frac{\rho_0}{\delta_s} \frac{p_i}{\rho_i^2} \nabla W_{is} = -m_i \frac{\rho_0}{\rho_i^2} p_i \sum_s \frac{1}{\delta_s} \nabla W_{is}$, and thus I obtain the following relation from Eq. (3.1), $\Delta \mathbf{u}_i$ and $\mathbf{F}_i^{\mathrm{p,solid}}$:

$$\frac{\rho_0^2}{\rho_i^2} \| \sum_s \frac{1}{\delta_s} \nabla W_{is} \|^2 p_i = \frac{\rho_i^* - \rho_0}{\Delta t^2}.$$

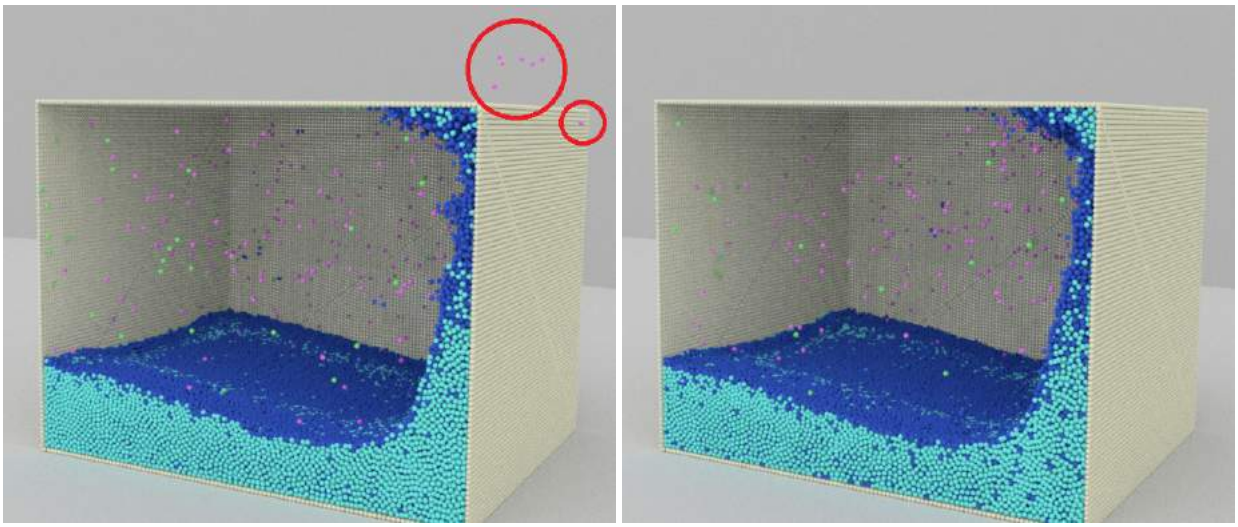


Figure 3.2: Dam break. Without the source term amplification, pressures can be underestimated, failing to prevent particle penetrations, as noted by red circles, (left), whereas with the source term amplification, particle penetrations are prevented (right).

Since $-\nabla^2 p_i^{\text{solid}}$ can be reformulated as $-\frac{\rho_i}{m_i}\nabla\cdot\left(\frac{m_i}{\rho_i}\nabla p_i^{\text{solid}}\right)$, which corresponds to the combination of the continuity equation and pressure force, I obtain $-\nabla^2 p_i^{\text{solid}} = \frac{\rho_0^2}{\rho_i^2}\|\sum_s \frac{1}{\delta_s}\nabla W_{is}\|^2 p_i$ and

$$\sum_j a_{ij}(p_i - p_j) + \alpha_i p_i = b_i, \tag{3.2}$$

where $\alpha_i = \frac{\rho_0^2}{\rho_i^2}\|\sum_s \frac{1}{\delta_s}\nabla W_{is}\|^2 \geq 0$ and $b_i = \frac{\rho_i^* - \rho_0}{\Delta t^2}$. Note that while non-negative α_i ensures the diagonal dominance of the system unlike (Takahashi et al., 2016), p_i can be underestimated leading to particle penetrations (see Figure 3.2 (left)).

To compensate the underestimation, I approximate the original solution by amplifying the source term (right hand side) as $\beta_i b_i$, with an amplification factor $\beta_i \geq 1$. Since pressures of the original and modified PPE can be computed by $\frac{b_i + \sum_j a_{ij}p_j}{\sum_j a_{ij}}$ and $\frac{\beta_i b_i + \sum_j a_{ij}p_j}{\sum_j a_{ij} + \alpha_i}$, respectively, pressure differences Δp_i are written as

$$\Delta p_i = \frac{b_i + \sum_j a_{ij}p_j}{\sum_j a_{ij}} - \frac{\beta_i b_i + \sum_j a_{ij}p_j}{\sum_j a_{ij} + \alpha_i}.$$

I can exactly solve $\Delta p_i = 0$ and obtain β_i as

$$\beta_i = \frac{b_i \sum_j a_{ij} + \alpha_i b_i + \alpha_i \sum_j a_{ij}p_j}{b_i \sum_j a_{ij}}.$$

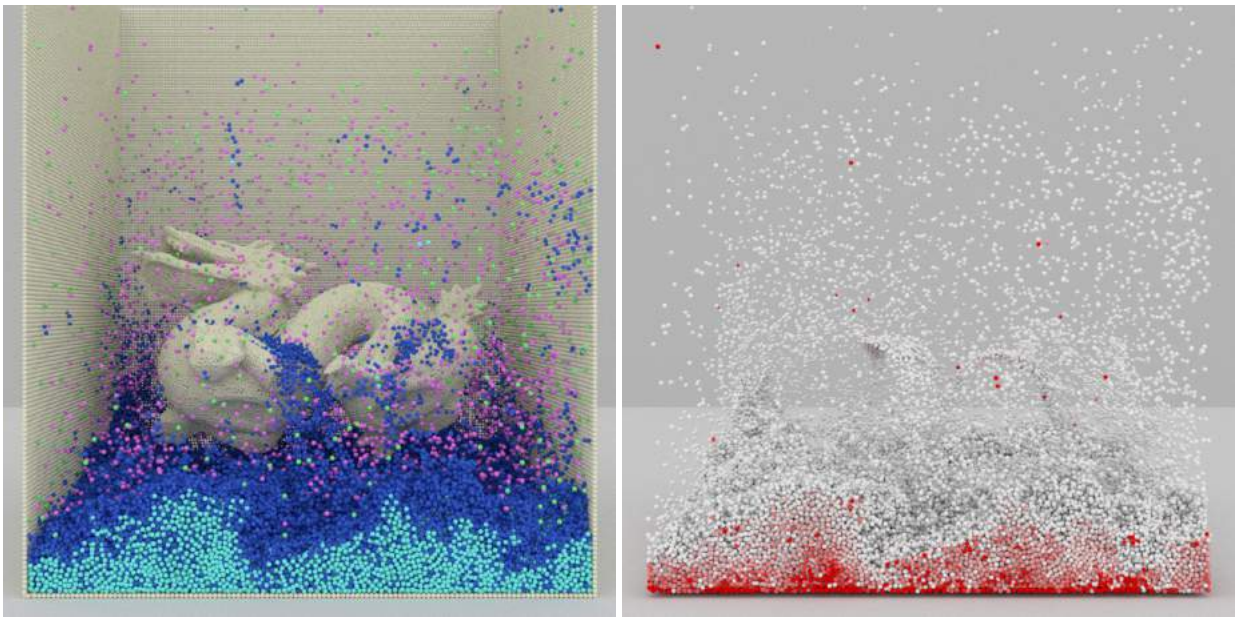


Figure 3.3: A cubed fluid dropped onto a solid dragon floating in the air. Particles are color coded based on the classification on the left, and pressure on the right, where white and red represent low and high pressures, respectively. With my method, particle pressures are definable regardless of the particle configurations.

Since physical values change only slightly between consecutive steps, and pressures of all particles are always definable with my method, I can assume $p_i \approx \tilde{p}_i$ ($\tilde{p}_i$: pressure at the previous step). Further assuming $p_i \approx p_j$ because of the nature of the PPE's solution, the amplified source term can be computed by

$$\beta_i b_i = \frac{\sum_j a_{ij} + \alpha_i}{\sum_j a_{ij}} b_i + \alpha_i \tilde{p}_i.$$

When $\sum_j a_{ij}$ is very small, $\beta_i b_i$ can be too large leading to infinitely large pressures. To avoid this, I clamp $\frac{\sum_j a_{ij} + \alpha_i}{\sum_j a_{ij}}$ with a limiting factor $\gamma \geq 1$ (I empirically found that $\gamma = 5.0$ works well) and finally obtain the PPE as

$$\sum_j a_{ij}(p_i - p_j) + \alpha_i p_i = \min\left(\gamma, \frac{\sum_j a_{ij} + \alpha_i}{\sum_j a_{ij}}\right) b_i + \alpha_i \tilde{p}_i. \tag{3.3}$$

For separated particles, β_i cannot be defined because they have no fluid neighbors (i.e., $\sum_j a_{ij} = 0$). In this case, I directly set their pressures without including these particles in the PPE (as Dirichlet boundary condition) based on Eq. (3.2) clamping negative values as $p_i = \max\left(0, \frac{b_i}{\alpha_i}\right)$.

My solid boundary handling method introduces $\alpha_i p_i$ into the left hand side of the PPE turning a Poisson equation to a Helmholtz equation when fluid particles are contacting with solid particles. This is numerically

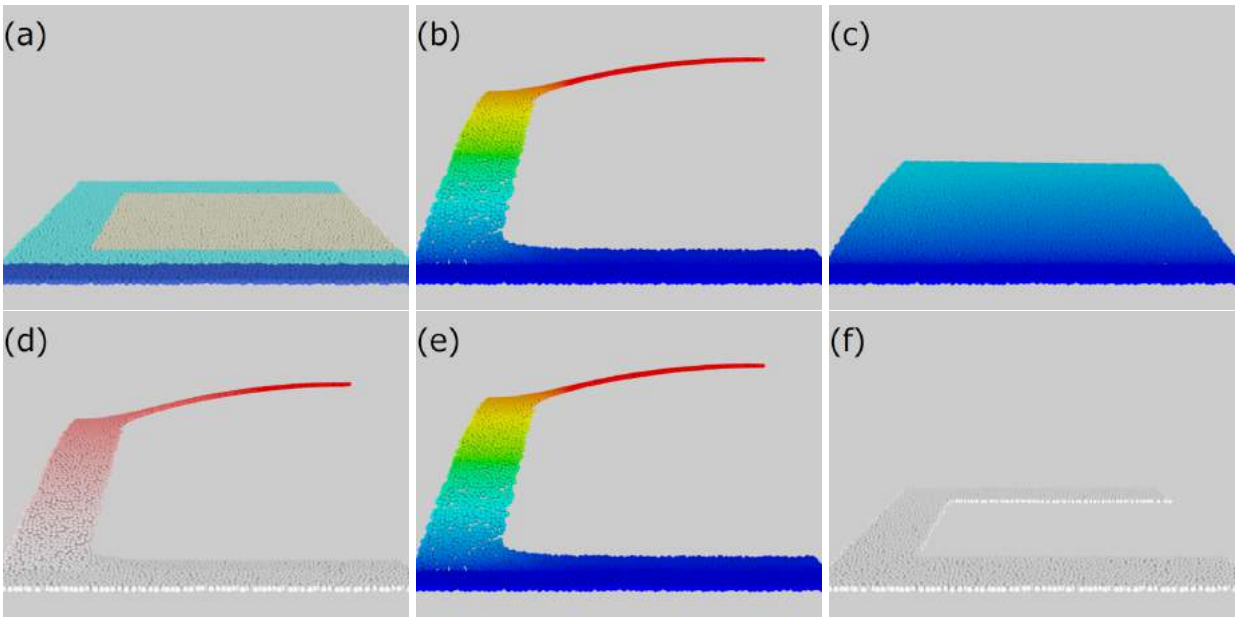


Figure 3.4: Comparisons on pressure accuracy with a square domain, where positive source terms are set on Poisson particles. In (b), (c), and (e), blue, green, and red represent low, middle, and high values, respectively, while in (d) and (f), white and red represent low and high values, respectively. In (b), (d), (e), and (f), Neumann particles are not visualized. (a) Scene setup. (b) Exact solution. (c) Solution obtained with the previous approach (Shao and Lo, 2003). (d) Solution difference (b) $-$ (c). (e) Solution obtained with my solid boundary handling. (f) Solution difference (b) $-$ (e).

similar to adding a Dirichlet boundary condition to the system, and thus makes the system solvable without satisfying the compatibility condition regardless of the particle configurations. When the exact solution cannot be globally defined, My method alters the solution based on the pressure force formulation minimizing the errors in regions, where the exact solution can be defined. As compared to (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003), my method constructs smaller systems and produces more accurate pressures. In addition, unlike these previous methods, my method can handle solid objects floating in the air (see Figure 3.3).

Figure 3.4 compares my solid handling method (I used an exactly computed p_i for $\tilde{p}_i$ in Eq. (3.3)) with a previous method (Koshizuka et al., 1996; Premoze et al., 2003; Shao and Lo, 2003) and an exact solution. Since it is not possible to obtain the exact solution in general fluid simulation scenarios as explained in Section 3.4.1, I experiment with a static 2D square domain, where an exact solution can be defined. The previous method that treats solid Neumann particles as Poisson particles connect Poisson and Dirichlet particles, newly generating shorter paths. Consequently, pressures are restricted to low values, and this

approach underestimates pressures (see (c) and (d)). On the other hand, my method can more accurately compute pressures with imperceptible errors only, as shown in (e) and (f).

3.5 Multilevel Particle-based Solver

In this section, I describe my multilevel solver, which is specifically designed for particle-based methods. For fundamentals of MG solvers, I refer to (Briggs et al., 2000; Trottenberg et al., 2000).

3.5.1 Hierarchical Structure Construction

Constructing the hierarchical structures for particle-based methods is difficult mainly because of irregular particle positions and changing particle neighbors (i.e., connectivities). For mesh coarsening, Müller (Müller, 2008) proposed to successively coarsen finer-levels preserving only representative particles, and Sacht et al. (Sacht et al., 2015) presented a method for constructing the hierarchy of surface meshes such that coarser levels are contained by their finer levels. However, these approaches are essentially designed for unstructured meshes with no frequent connectivity changes, and thus they are computationally expensive for particle-based methods. Additionally, unstructured fine-level meshes produce unstructured coarser levels, where the number of edges is likely to be large, thereby leading to higher smoothing and residual computation cost as compared to structured meshes. Moreover, unstructured meshes are not suitable for parallelization since we need to use a Jacobi smoother, which is less effective than Gauss-Seidel (GS) and Red-Black GS (RBGS) smoothers (Briggs et al., 2000; Trottenberg et al., 2000). Taking these factors into account, I employ a static Cartesian grid. To simplify boundary handling, I store pressures at the center of each cell.

3.5.2 Particle-Grid Correspondence

While the Cartesian grid has desirable properties for my method, unlike a carefully designed hierarchy, this simple choice of the regular grid introduces a new problem: solutions at the particle and grid levels are not consistent (see Figure 3.5), mainly because of the inconsistency of discretization at particle and grid levels, particle irregularities, kernel definitions, accuracy, and parameters. For example, when I use SPH to estimate physical values at both grid and particle levels, the estimated values cannot be consistent since grid points are not uniformly surrounded by neighbors at the grid level. If I use Finite Difference (FD) and SPH to estimate physical values at grid and particle levels, respectively, the estimated values can also be different

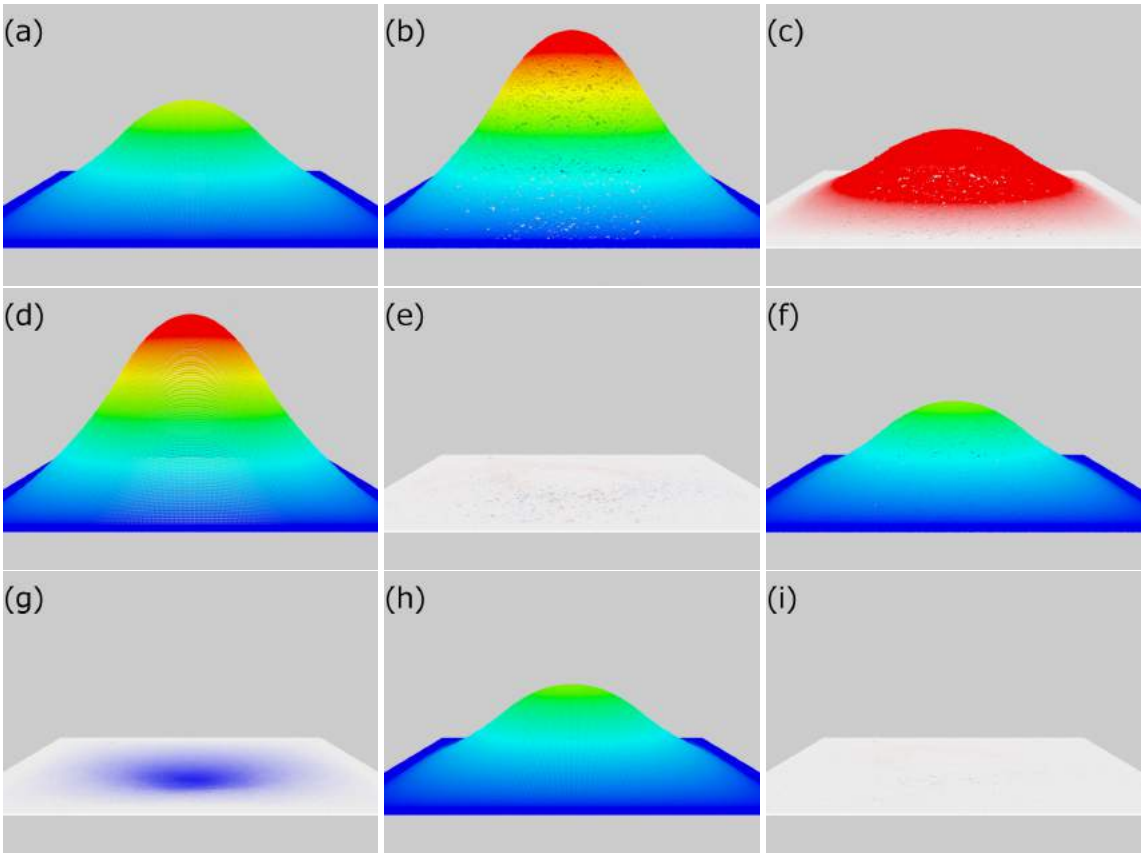


Figure 3.5: Correspondence check with a square domain, whose central regions have positive source terms, and whose edges are set as Dirichlet boundary condition. In (a), (b), (d), (f), and (h), blue, green, and red represent low, middle, and high values, respectively, while in (c), (e), (g), and (i), blue, white, and red represent low, middle, and high values, respectively. (a) Solution on the grid. (b) Solution on the particles, which is larger than (a). (c) Solution difference (b) $-$ (a). (d) Solution on the grid corrected with $\lambda^{\rm opt}$ approximating (b). (e) Solution difference (b) $-$ (d). (f) Solution on the particles, which is smaller than (a). (g) Solution difference (f) $-$ (a). (h) Solution on the grid corrected with $\lambda^{\rm opt}$ approximating (f). (i) Solution difference (f) $-$ (h).

because FD and SPH are different discretization methods. This problem is crucial, and if the solutions at the particle and grid levels are inconsistent, MG solvers converge slowly, stagnate, or diverge. To avoid these problems and hopefully achieve the optimal efficiency of MG solvers, solutions at the particle and grid levels need to be consistent.

I aim to establish the correspondence between solutions at particle and grid levels by modifying the solutions at the grid levels with a source term adjustment. Since solutions at the coarser grid levels always agree to the finest level up to the discretization accuracy, it is sufficient to establish the correspondence between the solutions at the particle and the finest grid levels. I use FD to estimate physical values at grid levels since the SPH-based estimation is more costly.

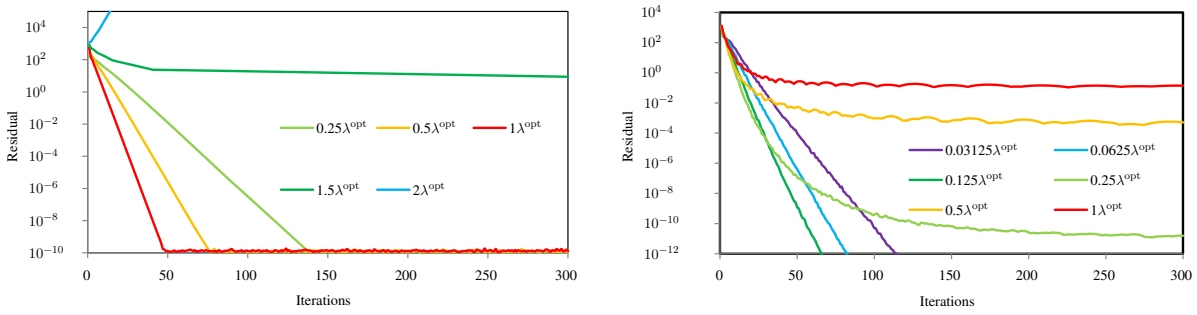


Figure 3.6: Convergence profile with different λ for V-cycle (left) and MGCG (right).

When the linear system at the particle and the finest grid levels are written as $\mathbf{A}^{\mathrm{P}}\mathbf{p}^{\mathrm{P}} = \mathbf{b}^{\mathrm{P}}$ and $\mathbf{A}^{\mathrm{G}}\mathbf{p}^{\mathrm{G}} = \mathbf{b}^{\mathrm{G}}$, respectively, the solutions are $\mathbf{p}^{\mathrm{P}} = (\mathbf{A}^{\mathrm{P}})^{-1}\mathbf{b}^{\mathrm{P}}$ and $\mathbf{p}^{\mathrm{G}} = (\mathbf{A}^{\mathrm{G}})^{-1}\mathbf{b}^{\mathrm{G}}$. Although it is optimal that each particle pressure p_i^P agrees to the interpolated grid pressure at the particle position p_i^G with a scaling parameter λ_i (i.e., $\mathbf{p}_i^P = \lambda_i \mathbf{p}_i^G$), changing connectivities makes achieving this impractical and complicated. Instead, taking into account that the overall pressure profiles are similar at the particle and grid levels, I use a global scaling factor λ obtaining a modified linear system as $\mathbf{A}^{\mathrm{G}}\tilde{\mathbf{p}}^{\mathrm{G}} = \lambda\mathbf{b}^{\mathrm{G}}$ with modified finest grid level pressure $\tilde{\mathbf{p}}^{\mathrm{G}}$ (i.e., $\tilde{\mathbf{p}}^{\mathrm{G}} = \lambda\mathbf{p}^{\mathrm{G}}$). Estimating the grid pressure at particle positions by trilinear interpolation (denoted by $\hat{I}_{\mathrm{G}}^{\mathrm{P}}$) and scaling λ, I can evaluate pressure error E by

$$E = \|\mathbf{p}^{\mathrm{P}} - \lambda\hat{I}_{\mathrm{G}}^{\mathrm{P}}\mathbf{p}^{\mathrm{G}}\|_2. \tag{3.4}$$

Note that Eq. (3.4) includes $\mathbf{p}^{\mathrm{P}}$ that I aim to get and changes over time, and thus I cannot directly minimize E in the simulation. Fortunately, an optimal λ^{opt} that minimizes E basically depends on the distributions of particles only and is not sensitive to the simulation scenarios. Thus, I precompute λ^{opt} with $\mathbf{p}^{\mathrm{P}}$ and $\mathbf{p}^{\mathrm{G}}$ obtained using CG, and determine λ by $\lambda = \gamma \min_t \lambda^t$ (γ: a tunable parameter) with which the MG solvers converge achieving a nearly optimal performance (see Figure 3.6). To achieve the correspondent solutions at particle and grid levels, I multiply λ when values at the particle level are transferred to the finest grid level (see Section 3.5.4).

Figure 3.5 illustrates the effect of established correspondence. While I obtain the same pressure profile with a fixed grid (see (a)), with different particle configurations, I would obtain pressures higher and lower than (a), as shown in (b) and (f), and their differences with respect to (a) are given in (c) and (g), respectively.

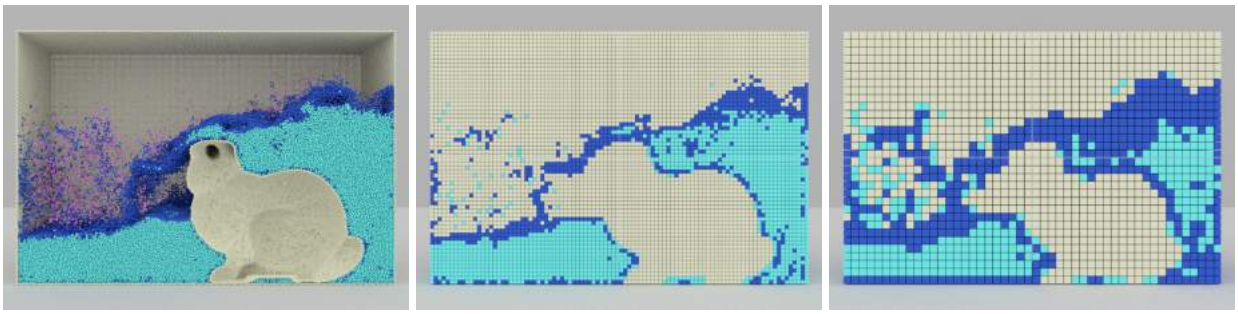


Figure 3.7: Cutaway views of fluid simulation with my coarsening scheme. Particle level, the finest grid level, and the second finest grid level from left to right.

My method modifies the pressure on the grid approximating (b) and (f), as shown in (d) and (h), and the differences can be largely corrected to 0, as shown in (e) and (i), respectively.

Figure 3.6 gives convergence profiles of V-cycle and MGCG for the scene shown in Figure 3.5. While $\lambda = \lambda^{\mathrm{opt}}$ achieved the best performance with V-cycle, interestingly the convergence of the MGCG with λ larger than $0.125\lambda^{\mathrm{opt}}$ became stagnant and achieved the best performance with $\lambda = 0.125\lambda^{\mathrm{opt}}$, presumably because preconditioning on irregular particle distributions causes overshoots and negatively affects the convergence of CG.

3.5.3 Linear System Construction

To construct linear systems at the finest grid level, I use the particle classification information. One possible approach is to generate signed distance functions for solid objects and fluid domains following the grid-based approaches (Bridson, 2015). However, this approach would erroneously compute fluid domains penetrating thin solids including volumetric objects whose only surfaces are sampled with particles as commonly done in particle-based methods (Akinci et al., 2012), and thus cannot approximate the particle level solution. To avoid this problem, I classify grid cells as follows. First, if there is at least one Dirichlet particle in a cell, I classify the cell as a Dirichlet cell (rendered as blue). Second, if there is at least one Poisson particle, I classify the cell as a Poisson cell (cyan). Otherwise, I classify the cell as a Neumann cell (beige). Since isolated and separated particles do not affect the solution of the system, I ignore these particles in my coarsening scheme. Although this approach introduces some discrepancies at the particle and the finest grid levels, Dirichlet boundary condition always remains at the finest grid level, and thus I can ensure the solvability of the system.

To obtain linear systems at coarser grid levels, I use the voxel-based approach proposed by (McAdams et al., 2010). Figure 3.7 illustrates cutaway views of fluid simulation with my coarsening scheme, at the particle, the finest grid, and the second finest grid levels. It is worth noting that I tested the cut-cell approach (Weber et al., 2015). However, the convergence was not improved, unlike the case of the grid-based simulations. This is presumably because the geometric consistency was already lost at the approximation of particles with the finest grid. I leave this issue as future work.

3.5.4 Restriction and Interpolation

While there are several ways to do restriction and interpolation, as previously proposed (McAdams et al., 2010; Weber et al., 2015), I need to satisfy $R_l^{l+1} = k(I_{l+1}^l)^T$ (R_l^{l+1}: restriction operator from level $l+1$ to l, I_{l+1}^l: interpolation operator from level l to $l+1$, and k: a constant), known as the Galerkin property (Briggs et al., 2000; Trottenberg et al., 2000), to use V-cycle as a preconditioner of CG. Among commonly used restriction and interpolation operations satisfying this condition are piecewise constant interpolation and linear interpolation. In (Dick et al., 2016), it is reported that the convergence rate is faster with linear interpolation than with piecewise interpolation, when the grid-based approaches are used. However, with particle-based methods, I did not observe the accelerated convergence due to the solution inconsistency between particles and grids, and the computational cost of the linear interpolation was much more expensive than the piecewise interpolation. Thus, I use the piecewise constant interpolation.

For the restriction from the particle level to the finest grid level, I compute the average, taking λ into account for correspondence, as $\phi_c = \lambda \frac{\sum_i \phi_i}{\sum_i}$, where ϕ denotes an arbitrary value, and i the index of particles in cell c. For the interpolation from the finest grid level to the particle level, I directly use the value in the cell for the particle as $\phi_i = \phi_c$. Note that I do not use the expensive trilinear interpolation $\hat{I}_{\mathrm{G}}^{\mathrm{P}}$ defined in Section 3.5.2.

3.5.5 Smoother

At the particle level, I use the weighted Jacobi method as a smoother to fully parallelize the pre- and post-smoothing operations, whereas at the grid levels, I use RBGS by taking advantage of the regular grid structures.

In some grid-based MG solvers, e.g., (Ferstl et al., 2014; Weber et al., 2015), CG is used to exactly solve the system at the coarsest level. In particle-based methods, however, since I terminate solver iterations

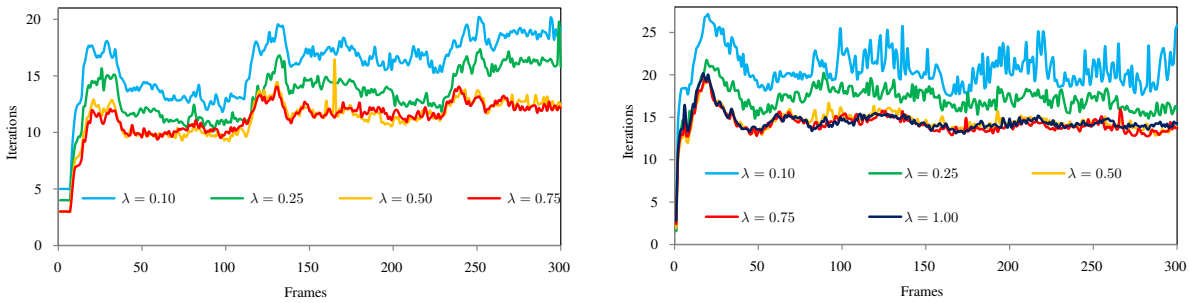


Figure 3.8: Iteration profiles for different values of λ in the two scenes (left for Figure 3.1 (left) and right for Figure 3.1 (middle)).

based on density error criteria (Solenthaler and Pajarola, 2009; Ihmsen et al., 2014a), which are much more moderate than residual criteria used in the Eulerian method (Bridson, 2015), I do not need to obtain the exact solution at the coarsest level, and thus use the multiple RBGS smoothing for efficiency.

3.5.6 Implementation

I define the grid width Δx at the finest grid level as $\Delta x = h$ since it is preferable to choose a grid width two times larger than particle distances (which are generally $0.5h$ when I set $h = 4r$, where r is the particle radius) (Briggs et al., 2000; Trottenberg et al., 2000). Additionally, this choice allows for the reuse of the uniform grid constructed for the neighbor search (Ihmsen et al., 2011). I did not achieve faster convergence with a smaller grid width due to the solution gaps between particles and grids, whereas I observed slower convergence with a larger grid width (this corresponds to the method of (Raveendran et al., 2011)) due to the less accurate approximation. While it is possible to use the multilevel method as a stand-alone solver, I prefer using it as a preconditioner for CG (i.e., as MGCG) to improve the robustness and efficiency (McAdams et al., 2010; Ferstl et al., 2014).

3.6 Results

I measured the performance on a machine with 24-core CPU and 256 GB RAM. For the simulation, I used a constant time-step based on the CFL condition, and the density deviation threshold was set to 0.01%. In the following, I used MGCG with 1 V-cycle preconditioning using 1 pre- and post- smoothing per iteration. Without using an appropriate scaling factor λ, my multilevel solver suffered from divergence or stagnation.

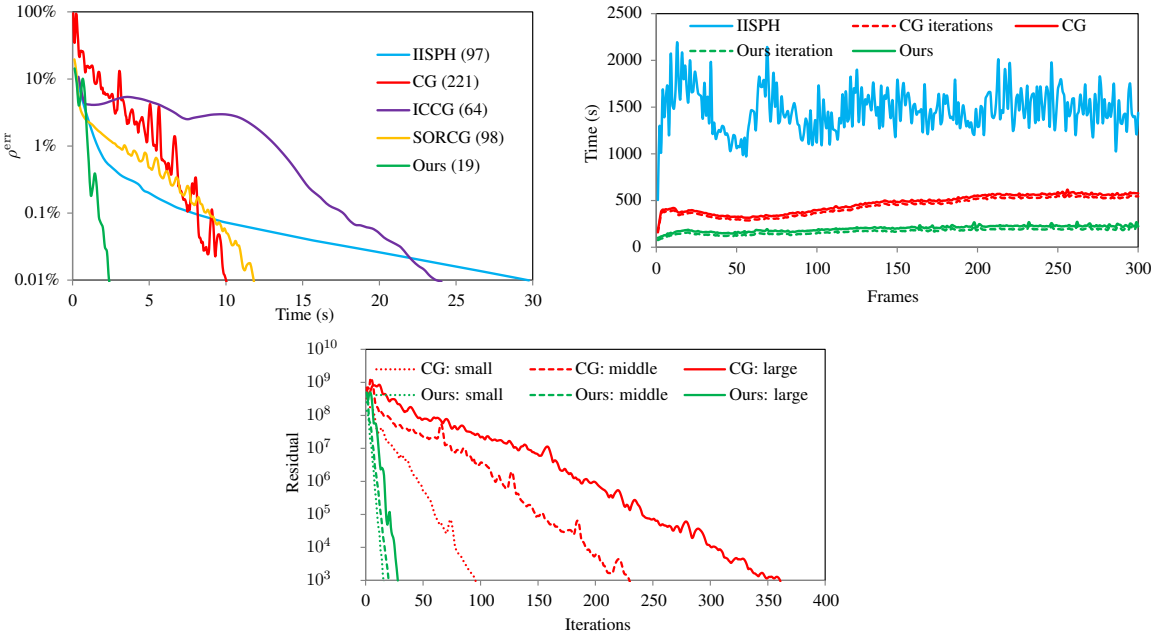


Figure 3.9: (Left) Density error (ρ^{err}) profile for my MGCG method, CG, ICCG, SORCG, and IISPH with respect to time. The number in the parentheses represents the number of iterations required to converge to a density error of less than 0.01%. (Middle) Performance profile for IISPH, ISPH with CG, and my method for the middle image of Figure 3.1. "CG iteration" and "Ours iteration" represent the computation time for the CG iterations only while "CG" and "Ours" includes the computation time for the system construction. My method outperforms IISPH and CG by a factor of 7.5 and 2.3, respectively. (Right) Residual profile of CG and mine for small, middle, and large scale scenarios with respect to iterations. The number of iterations for CG increases depending on the simulation scale, whereas my method requires almost the same number of iterations regardless of the simulation scale.

Influence of λ. To demonstrate the influence of λ, I tested several values of λ (0.10, 0.25, 0.50, 0.75, and 1.0) in the two scenes shown in Figure 3.1 (left) on a grid resolution of 96x64x96 with up to 822.4k particles and Figure 3.1 (middle) on a grid resolution of 96x64x64 with 1.0M particles. Figure 3.8 illustrates profiles of average iterations, where the profile of $\lambda = 1.0$ for Figure 3.1 (left) is not given since my MG solver did not converge. In my experiments, larger λ would cause failure of my solver, whereas smaller λ weakens the effect of the MG preconditioning slowing the convergence.

Convergence speed. I compared my MGCG solver with other solvers in the convergence speed, using the scene shown in Figure 3.1 (middle) with 3.4M particles on a grid of resolution 144x96x96. For this comparison, I used one of the state-of-the-art particle-based solver IISPH (Ihmsen et al., 2014a), and CG solvers commonly used in the particle-based methods: CG, Incomplete Cholesky CG (ICCG), and successive over-relaxation CG (SORCG). I used SOR in a Jacobi way (not as in Gauss-Seidel) for parallelization while I

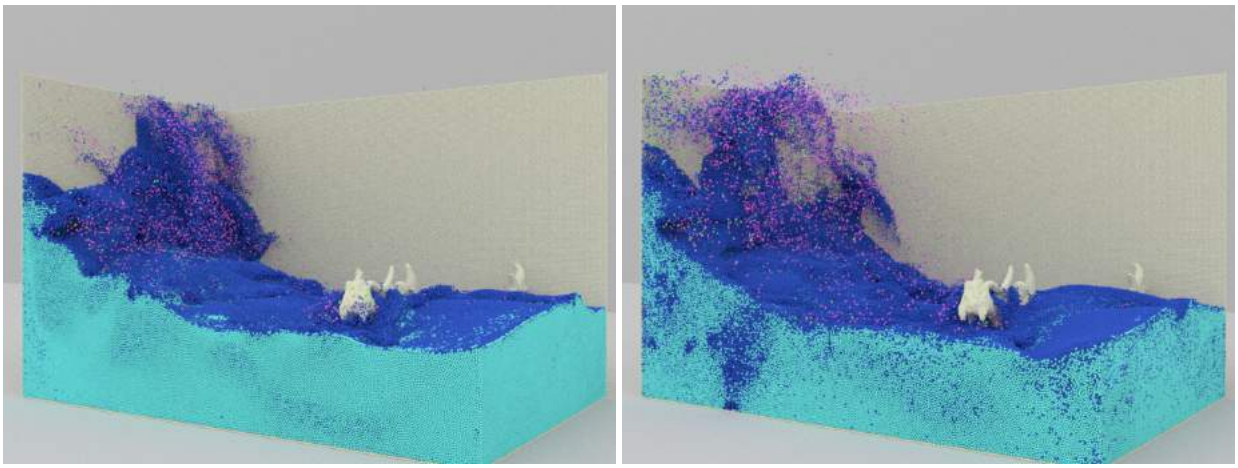


Figure 3.10: Visual comparison for my method, i.e., ISPH with MGCG (left) and IISPH (right). Both methods generate comparable fluid behaviors.

applied the IC preconditioner in a serial manner as the application of the IC is inherently serial. Figure 3.9 (left) illustrates a profile of the density error with respect to the computation time. Although IISPH converges faster at the early stage, the convergence becomes slower at the latter stage due to the use of Jacobi method. Consequently CG solvers become advantageous; in particular, my MGCG solver is 12x faster compared to IISPH. Though the IC and SOR preconditioners reduce the number of CG iterations, the preconditioning is costly and not effective enough to achieve a performance gain. As a result, CG is faster than ICCG and SORCG. On the other hand, my MG preconditioner can significantly reduce the CG iterations and can be efficiently applied. Consequently, my method can achieve a performance gain over CG by a factor of 4.0, which cannot be achieved with relatively simple IC and SOR preconditioners.

Overall performance. I compared my method (ISPH with MGCG) with previous methods (IISPH (Ihmsen et al., 2014a) and ISPH with CG) on the dam break scenario, where I used 3.4M fluid particles on a grid of resolution 144x96x96, as shown in Figure 3.10 (since ISPH with CG and MGCG generates essentially the same visual result, the result for CG is omitted). My method and IISPH generate comparable visual results. Figure 3.9 (middle) shows the performance on the pressure computation, where profiles for CG and MGCG iterations only (excluding the system construction) and the pressure solve with CG, MGCG, and IISPH are given (IISPH does not construct the system, and thus there is no system construction cost). My method outperforms IISPH and CG by a factor of 7.5 and 2.3 in the pressure solve, respectively, and achieves 2.6x better performance than CG in the iteration part.

Table 3.1: Performance comparisons with different time steps for IISPH and my method for 434.7k particles on a grid of resolution 72x48x48. l denotes Jacobi iteration for IISPH and CG iteration for my method. t^{p} denotes the average pressure solve time for one frame (including the system construction for my method), and t^{t} denotes the average total time for one frame. The best t^{p} and t^{t} are highlighted in red. My method outperforms IISPH by a factor of 6.3 in the pressure solve and 5.2 in the total time for their best t^{p} and t^{t}, respectively, even in a relatively small scenario.

	IISPH			My method		
$\Delta t(ms)$	l	$t^{\mathrm{p}}(s)$	$t^{\mathrm{t}}(s)$	l	$t^{\mathrm{p}}(s)$	$t^{\mathrm{t}}(s)$
2.08	13.15	45.97	62.56	15.71	34.97	51.33
4.16	30.66	53.80	62.23	12.63	15.33	23.70
8.32	80.46	73.43	78.01	12.22	7.31	11.88

Time step effect. I compared my method with IISPH using different time steps on the dam break scenario, where I used 434.7k particles on a grid of resolution 72x48x48, and summarized their performance in Table 3.1. With IISPH, using larger time steps can significantly increase the Jacobi iterations for convergence leading to more computational cost. On the other hand, with my method, the number of CG iterations does not increase even though larger time steps are used because of the fact that my MGCG solver can effectively handle ill-conditioned systems. This feature of my solver makes it easier to choose appropriate time steps since I can optimize the performance by just choosing the largest time step possible. In general, it is preferable to use larger time steps to accelerate the entire simulation as long as the simulation is stable. With this general rule (i.e., with $\Delta t = 8.32$ (ms)), the gain of my method over IISPH is 10.0x in the pressure solve and 6.6x in the total time.

Scalability. To demonstrate the effectiveness of my method on the scalability, I performed the dam break scenario with three different scales: 123.5k particles with the grid resolution 48x32x32, 1.0M particles with the grid resolution 96x64x64, and 3.4M particles with the grid resolution 144x96x96. For this comparison, I used the residual of the PPE since the positive density error ρ^{err} (commonly used in the particle-based methods) depends on time steps (Takahashi et al., 2016) and changes according to the simulation scales, and thus I cannot compare different scale scenarios under a fair condition. Figure 3.9 (right) demonstrates the profile of residual over iterations to show the scalability of the solvers. With CG, the number of iterations increases significantly as the simulation scale becomes larger. On the other hand, my method requires almost the same number of iterations regardless of the simulation scale, as expected from the theory of the MG solvers. Since my method scales well with respect to the number of particles unlike CG, I believe that my method will be more advantageous with larger-scale scenarios.

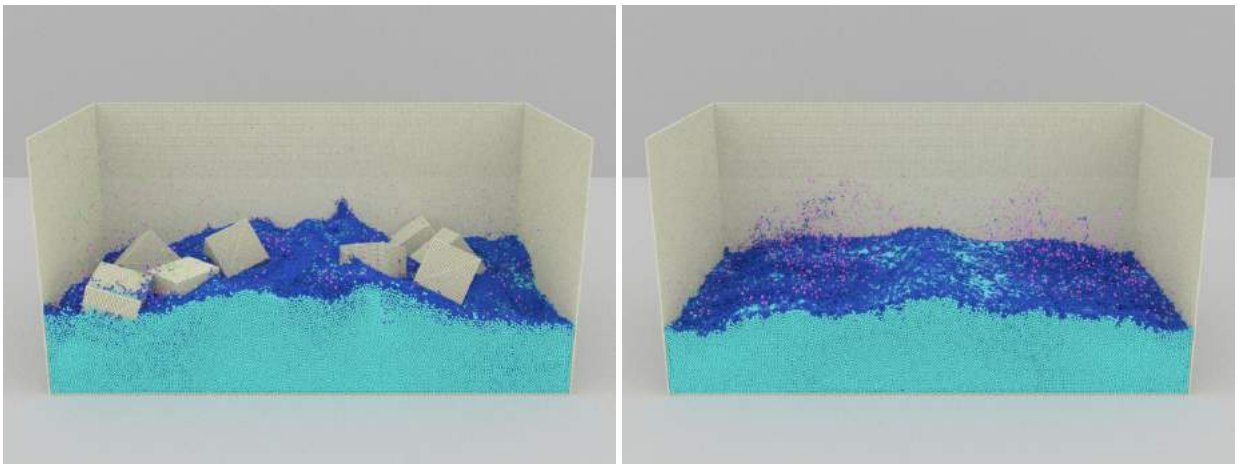


Figure 3.11: Double dam break with (left) and without (right) two-way coupled solid objects. Both scenarios require similar iteration counts for convergence (see Figure 3.12).

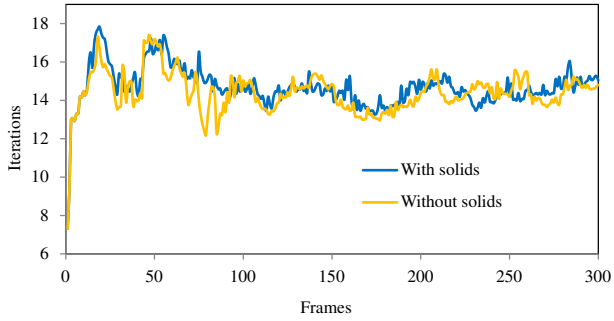


Figure 3.12: Iteration profile for Figure 3.11. Regardless of the two-way coupled solids, the number of iterations is comparable.

Solid interaction. Figure 3.11 (left) demonstrates that my method is capable of handling two-way coupled solid objects. Additionally, I compare the number of required iterations in Figure 3.12 for the double dam break scenarios with and without solid cubes (see Figure 3.11), and show that the number of iterations is comparable regardless of the additional complexity introduced by the two-way coupled solid cubes. My method is general and can simulate further complex scenarios, such as one-way coupling of fluids with fast moving solid bunnies (Figure 3.1 (left)) and two-way coupling of solids in multiphase flows (Figure 3.1 (right)).

3.7 Discussions and Limitations

My particle-based method uses auxiliary grid structures, and thus my method can be considered as a hybrid approach, such as FLIP. However, since the grid structures are used to merely accelerate the pressure solve on particles, my method essentially differs from FLIP. It is worth noting that my multilevel solver is an error correction approach same as other MG, and the communication between particles and grids does not introduce any errors to the final converged solution at the particle level. Because of the auxiliary grid structures, it may seem that my method weakens advantages of particle-based methods. However, I can handle collision detection and resolution at the particle level allowing for the natural coupling of fluids with solid objects (which are commonly described in the Lagrangian manner), as demonstrated in Figures 3.1 (left and right) and 3.11. In addition, the grid structures are merely auxiliary structures. Thus, I do not need to adapt the grid structures, e.g., to free surfaces and solid boundaries, unlike the grid-based approaches, allowing for quite simple and fast hierarchy construction.

Although the optimal complexity of the MG solvers is $O(N)$ with the number of unknowns N, the number of iterations slightly increases with my method as N increases. One factor for this non-optimal complexity is due to the solution inconsistency between the particle and grid levels caused by the essentially different discretization methods and heuristically determined scaling factor λ. These would be addressed by using consistent discretization methods and more accurately estimating the optimal λ.

My MGCG method can outperform IISPH in certain scenarios. However, IISPH can be advantageous when simulations are performed under low-resolution with a soft density constraint since IISPH with Jacobi method converges faster at the early stage, and we may not be able to benefit from the fast convergence of CG at the latter stage.

To derive my solid boundary handling formulation, I assume that pressure changes over time and space are negligible between consecutive simulation steps, and these assumptions are used in the source term computation only. Since pressures are globally computed by solving the PPE, slight value changes in the source term do not significantly affect the resulting pressures. The resulting pressures are still smooth and thus do not introduce stability issues into the simulation. This fact can also be applied to the clamping of the source term. In the grid-based simulation literature, the source term modification is effectively used to compensate fluid volumes (Kim et al., 2007) and to simulate compressible fluids (Feldman et al., 2003) without stability issues.

3.8 Conclusion and Future Work

I proposed a new multilevel solver for particle-based fluids. My method constructs the hierarchy based on the Cartesian grid, establishes the correspondence between solutions at particle and grid levels, and coarsens simulation elements taking boundary conditions into account. In addition, I proposed a solid boundary handling method that ensures the solvability of the PPE without increasing the size of the system and computational cost. I demonstrated that my method can be significantly faster than IISPH, and its cost scales nearly linearly unlike previous particle-based solvers.

There are several promising future research directions. Since my method is massively parallelizable, implementing the algorithm on a GPU (Chentanez and Müller, 2011; Chentanez and Müller, 2012) is a natural extension of my method. In the same way as (Ferstl et al., 2014; Dick et al., 2016), introducing the cell duplication technique would be effective to improve the convergence rate. Since my multilevel solver can better handle ill-conditioned systems, aggressively using larger time steps would be beneficial. Considering that MG solvers use coarser levels, applying coarse grid approaches (Lentine et al., 2010; Edwards and Bridson, 2014) to particle-based methods would be interesting. Although it is known that geometric MG is generally faster than algebraic MG (Briggs et al., 2000), it would be worth comparing their performance. Particularly, the smoothed aggregation technique (Tamstorf et al., 2015) would be a promising choice.

Acknowledgements

This work is supported in part by JASSO for Study Abroad, U.S. National Science Foundation, and UNC Arts and Sciences Foundation. I thank anonymous reviewers for their valuable suggestions and comments which help improving the exposition of the paper.

CHAPTER 4: Implicit Formulation for SPH-based Viscous Fluids

4.1 Introduction

Smoothed Particle Hydrodynamics (SPH) is becoming increasingly popular for simulating fluids because of its attractive features including automatic conservation of mass, implicit tracking of surfaces with frequent topology changes, and no need for grid structures or meshes. SPH has been developed in various directions, e.g., enforcing fluid incompressibility (Becker and Teschner, 2007b; Solenthaler and Pajarola, 2009; Ihmsen et al., 2014a), handling fluid-fluid and fluid-solid interactions (Müller et al., 2005; Solenthaler et al., 2007; Solenthaler and Pajarola, 2008; Becker et al., 2009; Akinci et al., 2012; He et al., 2012b; Ren et al., 2014), and improving computational efficiency and saving memory usage (Adams et al., 2007; Ihmsen et al., 2011; Solenthaler and Gross, 2011; Orthmann and Kolb, 2012), and is recognized as a state-of-the-art fluid solver in computer graphics (Ihmsen et al., 2014b).

Over the past decades, various SPH methods have been proposed and used for a variety of fluid effects in the literature. However, most of these SPH methods assume that fluid is inviscid or slightly viscous; thus an effective SPH method that can simulate highly viscous fluids has not yet been established although we see various viscous materials (e.g., honey, caramel sauce, melted chocolate, lava, machinery oils, and bodily fluids) and their characteristic behaviors on a daily basis. There are two main reasons for this; First, previous SPH methods, e.g., (Müller et al., 2003, 2005; Solenthaler et al., 2007), drop off-diagonal components of

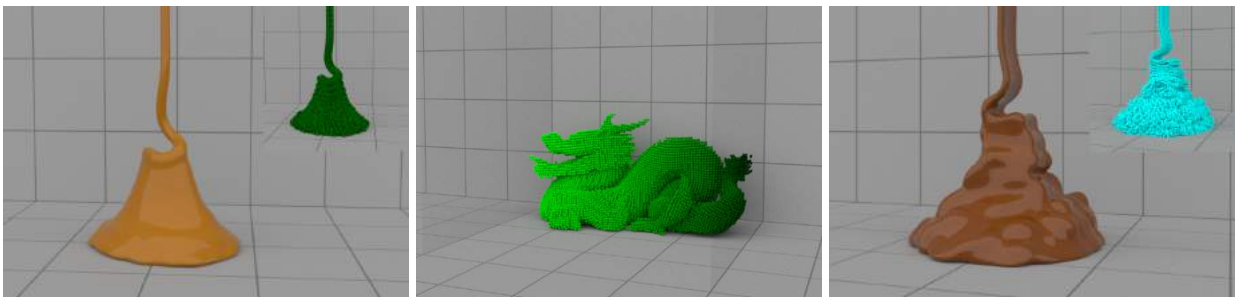


Figure 4.1: Viscous fluids simulated with my implicit formulation. Left to right: caramel sauce coiling with a particle view in the inset; a dragon consisting of particles with different viscosities; melted chocolate buckling with a particle view in the inset.

viscous stress tensor to simplify the viscosity term and consequently fail to generate rotational viscous fluid behaviors, such as coiling and buckling, due to neglect of the boundary condition on free surfaces, while leading to inaccurate handling of variable viscosity. This is also true for XSPH (Monaghan, 1989) and artificial viscosity (Monaghan, 1992) as they are essentially same as the Laplacian form in simplifying the viscosity term. Since the simplified term is described by Laplacian operator, in this chapter I refer to this term as *Laplacian form of viscosity* and the original, unsimplified term as *full form of viscosity*. Second, previous SPH methods, e.g., (Paiva et al., 2006; Andrade et al., 2014), suffer from a restriction on available time steps because of their explicit viscosity integration schemes. Andrade et al. (Andrade et al., 2014) proposed a condition on time steps $\Delta t \leq 0.1\frac{\rho h^2}{8\mu}$ (Δt: time step, ρ: fluid density, h: kernel radius, and μ: dynamic viscosity) to perform numerically stable simulations of viscous fluids with the full form, and this condition makes it difficult to simulate viscous fluids within a reasonable time when higher viscosity and resolutions (smaller kernel radii) are used.

To address the two problems above, I propose a new SPH-based method that uses implicit viscosity integration for the full form for robustly simulating highly viscous fluids. My method offers the following advantages:

- It is efficient, allowing for use of larger time steps and finer spatial resolutions than explicit integration.
- It is robust and stable, even with large time steps and high viscosities.
- It can generate coiling and buckling phenomena and handle variable viscosity.

I exploit the *variational principle* that automatically enforces the boundary condition on free surfaces to derive my implicit formulation, constructing a sparse linear system with a symmetric positive definite matrix. To efficiently solve the linear system, I also propose a novel method for extracting coefficients of the matrix that includes contributions from first-ring neighbor particles and second-ring neighbor particles (neighbor particles' neighbor particles). Figure 4.1 demonstrates viscous fluids, simulated using my implicit formulation.

In Eulerian methods, I can easily discretize the full form (divergence of Jacobian of velocity) at a time using finite difference due to the staggered stress arrangement as in (Batty and Bridson, 2008). On the other hand, in SPH, I first need to compute Jacobian of velocity, and then compute divergence of Jacobian of velocity, separately applying SPH formulations to both steps. Although computing these two steps is easy for

explicit integration (Paiva et al., 2006; Andrade et al., 2014), significant complexity is involved with implicit integration, and I need to take into account contributions from both of first-ring and second-ring neighbors to construct a linear system. Because of this complexity unique to SPH discretization and derivation of the implicit formulation, no work has been proposed regardless of the apparent simplicity of the idea in adopting implicit integration and demand for robust SPH-based simulators. To the best of my knowledge, my method is the first SPH method that uses implicit integration for the full form of viscosity, and is also the first method that extracts matrix coefficients contributed by second-ring neighbors.

4.2 Related Work

Eulerian viscous fluids. Carlson et al. (Carlson et al., 2002) first enabled stable simulations of highly viscous fluids with free surfaces by solving the Laplacian form of viscosity using implicit integration. Later, Rasmussen et al. (Rasmussen et al., 2004) proposed an implicit-explicit scheme for the full form of viscosity to correctly handle variable viscosity at the expense of numerical stability. Batty and Bridson (Batty and Bridson, 2008) proposed a fully implicit viscosity integration scheme for the full form, making it possible to take larger time steps, handle variable viscosity, and generate coiling and buckling. This method was extended by Batty and Houston (Batty and Houston, 2011) for an adaptive tetrahedral fluid simulator. In the work of Stomakhin et al. (Stomakhin et al., 2014), a viscosity term was also solved using implicit integration in the framework of Material Point Method.

Lagrangian viscous fluids. I categorize Lagrangian methods into five groups: Lagrangian Finite Element Methods (Lagrangian FEM), dimensionally reduced discrete methods, spring-based methods, deformation-based methods, and SPH methods. My method belongs to the SPH methods.

Lagrangian FEM has been used to accurately simulate viscous fluids, and various developments have been done in the literature. Bargteil et al. (Bargteil et al., 2007) proposed an efficient remeshing method to reduce the cost of time-consuming remeshing process, and Wojtan and Turk (Wojtan and Turk, 2008) improved the remeshing method of Bargteil et al. (Bargteil et al., 2007). Wicke et al. (Wicke et al., 2010) proposed a local remeshing method to keep the number of tetrahedra small. Clausen et al. (Clausen et al., 2013) proposed a Lagrangian FEM that can handle elastic, plastic, and fluid materials in a unified manner.

To accurately simulate viscous threads and sheets, Bergou et al. (Bergou et al., 2010) and Batty et al. (Batty et al., 2012) proposed dimensionally reduced discrete methods and generated coiling and buckling, respectively, limiting the dimension of materials that they can simulate.

Spring-based methods have been used because of its conceptual simplicity. Miller and Pearce (Miller and Pearce, 1989) and Terzopoulos et al. (Terzopoulos et al., 1991) proposed a spring-based model that computes repulsion and attraction forces between particles. Clavet et al. (Clavet et al., 2005) extended this model to simulate materials that exhibit elasticity, viscosity, and plasticity. Takahashi et al. (Takahashi et al., 2014) also simulated such materials in a unified framework of Position-based dynamics.

Gerszewski et al. (Gerszewski et al., 2009) proposed a deformation-based method that approximates motions of neighbor particles based on deformations of particle configurations for reproducing elastoplastic materials. Their method was extended by Zhou et al. (Zhou et al., 2013) to improve its numerical stability using implicit integration and by Jones et al. (Jones et al., 2014) to handle varying mass materials.

Desbrun and Gascuel (Desbrun and Gascuel, 1996) used SPH to simulate viscous materials. Müller et al. (Müller et al., 2003) proposed the Laplacian form of viscosity and simulated slightly viscous fluid. The Laplacian form was also used in (Müller et al., 2005; Solenthaler et al., 2007). Paiva et al. (Paiva et al., 2006) proposed the full form of viscosity and accurately simulated viscous fluids. The full form was also used in the work of Andrade et al. (Andrade et al., 2014). Rafiee et al. (Rafiee et al., 2007) presented a method based on a Maxwell model to simulate coiling of viscoelastic fluids. Dagenais et al. (Dagenais et al., 2012) simulated viscous fluid motions by adding extra forces that move particles to their original positions. In Astrophysics, Monaghan (Monaghan, 1997) and Laibe and Price (Laibe and Price, 2012) used pair-wise implicit formulations for artificial viscosity. Their methods consider only pair-wise particles for implicit integration, and thus can be numerically unstable. In Computational Mechanics, Fan et al. (Fan et al., 2010) proposed an implicit scheme for simulating viscous fluids using SPH. However, they computed viscous stress with gradient of velocity, not Jacobian of velocity, and hence their method is essentially equivalent to a method that uses the Laplacian form.

Paiva et al. (Paiva et al., 2006) and Andrade et al. (Andrade et al., 2014) solved the full form of viscosity. However, my method differs from theirs in that theirs used explicit integration whereas mine uses implicit integration to improve the robustness of the simulation and enable use of larger time steps with higher viscosities and finer spatial resolutions than these methods.

4.3 Fundamentals for Simulating Viscous Fluids

Formulations. I aim to simulate incompressible, highly viscous fluids using SPH. In the Lagrangian setting, the Navier-Stokes equations for particle i can be described as

$$\rho_i \frac{d\mathbf{u}_i}{dt} = -\nabla p_i + \nabla \cdot \mathbf{s}_i + \frac{\rho_i}{m}\mathbf{F}_i^{\text{ext}}, \tag{4.1}$$

$$\mathbf{s}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T\right), \tag{4.2}$$

where ρ_i denotes density of particle i, t time, $\mathbf{u}_i = [u_i, v_i, w_i]^T$ velocity, p_i pressure, $\mathbf{s}_i$ viscous stress tensor, m mass (I use a constant mass for all particles), $\mathbf{F}_i^{\text{ext}}$ external force, and μ_i dynamic viscosity. The combination of the second term on the right side in Eq. (4.1), and Eq. (4.2) is the full form of viscosity and is required to handle variable viscosity (Rasmussen et al., 2004) and rotational behaviors (Batty and Bridson, 2008). I separate the terms in Eq. (4.1) to independently solve them, taking the standard approach of operator splitting as with Eulerian methods (Batty and Bridson, 2008; Batty and Houston, 2011), and enforce fluid incompressibility using a particle-based incompressible fluid solver, e.g., (Solenthaler and Pajarola, 2009; Ihmsen et al., 2014a).

To generate rotational behaviors by solving Eq. (4.1), I need to consider the boundary condition that there is no traction on free surfaces (Batty and Bridson, 2008). In other words, as a boundary condition for Eq. (4.1), I must satisfy $(-p_i\mathbf{I} + \mathbf{s}_i)\mathbf{n}_i = 0$ ($\mathbf{n}_i$: normal to the free surface). As with (Batty and Bridson, 2008; Stomakhin et al., 2014), I decouple these terms and independently enforce the boundary condition for pressure $-p_i\mathbf{I}\mathbf{n}_i = 0$ (I enforce this by setting pressures of surface particles to 0), and for viscosity $\mathbf{s}_i\mathbf{n}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T\right)\mathbf{n}_i = 0$. Since my method is based on the variational principle, the boundary condition for viscosity is automatically enforced by solving Eq. (4.1) (see e.g., (Batty and Bridson, 2008; Stomakhin et al., 2014)).

Algorithm. First, I apply only external force $\mathbf{F}_i^{\text{ext}}$ and obtain first intermediate velocity $\mathbf{u}_i^*$. Then, I solve viscosity using my implicit formulation (see § 4.4) and obtain second intermediate velocity $\mathbf{u}_i^{**}$ with intermediate viscous stress tensor $\mathbf{s}_i^{**}$:

$$\mathbf{u}_i^{**} = \mathbf{u}_i^* + \frac{\Delta t}{\rho_i}\nabla \cdot \mathbf{s}_i^{**}, \tag{4.3}$$

$$\mathbf{s}_i^{**} = \mu_i \left(\nabla \mathbf{u}_i^{**} + (\nabla \mathbf{u}_i^{**})^T\right). \tag{4.4}$$

Next, to enforce fluid incompressibility, I compute pressure p_i using a particle-based fluid solver with the boundary condition for pressure, and compute pressure force $\mathbf{F}_i^p$ (Monaghan, 1992). Finally, particle velocity $\mathbf{u}_i$ and position $\mathbf{x}_i = [x_i, y_i, z_i]^T$ are integrated using Euler-Cromer scheme. I summarize a full procedure of my method in Algorithm 3.

Algorithm 3 Procedure of my method

// j: neighbor particle of i
// W_{ij}: kernel with a kernel radius h
for all particle i **do**
    find neighbor particles
for all particle i **do**
    apply external force $\mathbf{u}_i^* = \mathbf{u}_i^t + \Delta t \mathbf{F}_i^{\text{ext}}/m$
for all particle i **do**
    solve viscosity using Eqs. (4.3) and (4.4) // § 4.4
for all particle i **do**
    compute p_i using a particle-based fluid solver
for all particle i **do**
    compute $\mathbf{F}_i^p = -m^2 \sum_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}$
for all particle i **do**
    integrate particle velocity $\mathbf{u}_i^{t+1} = \mathbf{u}_i^{**} + \Delta t \mathbf{F}_i^p/m$
    integrate particle position $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \mathbf{u}_i^{t+1}$

Unlike (Batty and Bridson, 2008) that enforces fluid incompressibility twice, I enforce fluid incompressibility once in one simulation step as in (Batty and Houston, 2011) because my method can sufficiently maintain incompressibility over all simulation steps (when fluid incompressibility was enforced with a tolerable density error of 0.1%, the maximum density deviation was lower than 0.5%), and I observed indiscernible differences between enforcing incompressibility once and twice.

4.4 Implicit Formulation for Full Form of Viscosity

In this section, I first describe how to solve the viscosity term using implicit integration in the SPH framework while constructing a linear system (§ 4.4.1). Next, I explain sparsity of the coefficient matrix (§ 4.4.2), and my solver and coefficient extraction method for the system (§ 4.4.3). Then, I give implementation details and show my algorithm for solving my implicit formulation (§ 4.4.4).

4.4.1 Implicit Integration for Full Form of Viscosity

I derive an SPH-based implicit viscosity formulation from Eqs. (4.3) and (4.4). Hereafter, I drop the symbol $**$ for readability.

Since the boundary condition for viscosity is automatically enforced because of the variational principle (Batty and Bridson, 2008; Stomakhin et al., 2014), I directly discretize Eqs. (4.3) and (4.4) using implicit integration in the SPH framework:

$$\mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left(\frac{\mathbf{s}_i}{\rho_i^2} + \frac{\mathbf{s}_j}{\rho_j^2} \right) \nabla W_{ij}, \tag{4.5}$$

$$\mathbf{s}_i = \mu_i \sum_j V_j \left((\mathbf{u}_j - \mathbf{u}_i) \nabla W_{ij}^T + \nabla W_{ij} (\mathbf{u}_j - \mathbf{u}_i)^T \right). \tag{4.6}$$

By substituting $\mathbf{s}_i$ in Eq. (4.6) into Eq. (4.5) and arranging the terms in these equations, I obtain an implicit formulation:

$$\mathbf{u}_i + \hat{m} \sum_j \left(\hat{\mu}_i \mathbf{Q}_{ij} + \hat{\mu}_j \mathbf{Q}_{jk} \right) \nabla W_{ij} = \mathbf{u}_i^*, \tag{4.7}$$

$$\mathbf{Q}_{ij} = \begin{bmatrix} 2\sum_j a_{ij,x} u_{ij} & q_{ij,xy} & q_{ij,xz} \\ q_{ij,xy} & 2\sum_j a_{ij,y} v_{ij} & q_{ij,yz} \\ q_{ij,xz} & q_{ij,yz} & 2\sum_j a_{ij,z} w_{ij} \end{bmatrix}, \tag{4.8}$$

$$q_{ij,xy} = \sum_j \left(a_{ij,y} u_{ij} + a_{ij,x} v_{ij} \right), q_{ij,xz} = \sum_j \left(a_{ij,z} u_{ij} + a_{ij,x} w_{ij} \right),$$
$$q_{ij,yz} = \sum_j \left(a_{ij,z} v_{ij} + a_{ij,y} w_{ij} \right),$$

where $\hat{m} = m\Delta t$, $\hat{\mu}_i = \mu_i / \rho_i^2$, k is a neighbor particle of j, $\mathbf{a}_{ij} = [a_{ij,x}, a_{ij,y}, a_{ij,z}]^T = V_j \nabla W_{ij} = V_j [\nabla W_{ij,x}, \nabla W_{ij,y}, \nabla W_{ij,z}]^T$, $u_{ij} = u_i - u_j$, $v_{ij} = v_i - v_j$, and $w_{ij} = w_i - w_j$. $\mathbf{Q}_{ij}$ is symmetrical due to the symmetrical property of $\mathbf{s}_i$. This implicit formulation Eq. (4.7) is a linear system and can be rewritten in a matrix form as $\mathbf{CU} = \mathbf{U}^*$ ($\mathbf{C}$ is a coefficient matrix and $\mathbf{U} = [\ldots, u_i, v_i, w_i, \ldots]^T$). Let N denote the number of fluid particles, the size of $\mathbf{C}$ is $3N \times 3N$, and that of $\mathbf{U}$ is $3N \times 1$. I assign serial numbers (id) to particles to specify locations in a coefficient matrix.

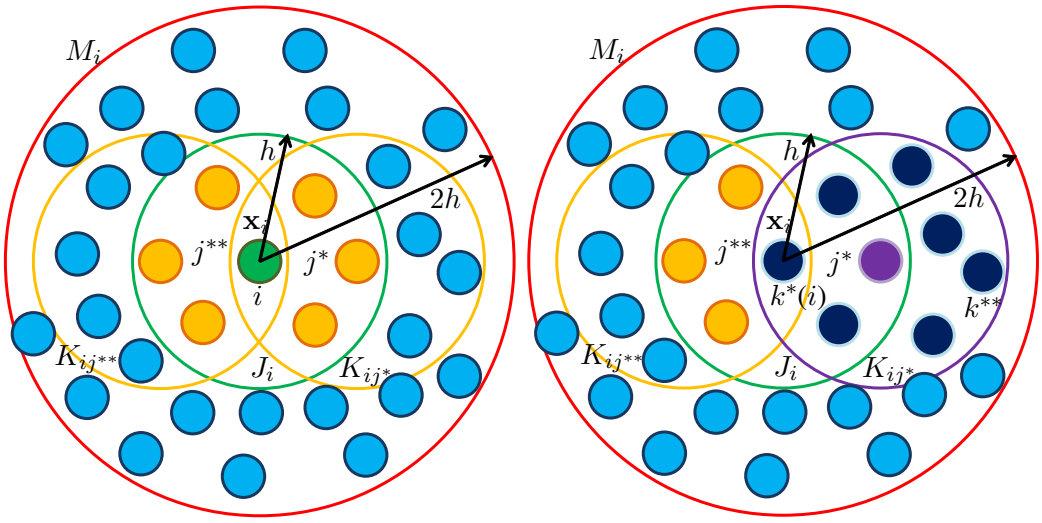


Figure 4.2: Illustration for first- and second-ring neighbors. Left: particle i (green) directly interacts with i's first-ring neighbors J_i (particle j colored in orange) inside of i's kernel sphere with its radius h shown as a green circle. Right: particle j^* (purple) has neighbor particles (within j^*'s kernel sphere shown as a purple circle), which are second-ring neighbors K_{ij^*} for particle i. Possible second-ring neighbors for particle i exist within the largest Minkowski sum (M_i) of J_i and K_{ij}, which is a sphere shown as a red circle, and M_i's center and radius are $\mathbf{x}_i$ and $2h$, respectively.

4.4.2 Sparsity of Coefficient Matrix

In SPH, particle i interacts with its neighbor particles within its kernel radius, namely inside of its kernel sphere. Since my formulation requires viscous stress of particles i and j to compute velocity update for particle i using Eq. (4.5), not only i's neighbors J_i (a set of particle j) but j's neighbors K_{ij} (a set of particle k) must be taken into account. In short, I need to include contributions from first-ring neighbors J_i and second-ring neighbors K_{ij} to compute i's velocity update and construct a linear system. This setup is illustrated in Figure 4.2. Assuming that particles i and j are spherically surrounded by others, i and j generally have 30-40 first-ring neighbors within their kernel radius h (Solenthaler and Pajarola, 2009). Since particle j can exist anywhere within i's kernel sphere, the total number of second-ring neighbors of particle i without overlaps can be 240-320. This is because the maximum (total) number of second-ring neighbors of i without overlaps is smaller than the number of particles within the largest Minkowski sum of i's kernel sphere and js' kernel spheres, and the largest Minkowski sum M_i is a sphere whose center is $\mathbf{x}_i$ and radius is $2h$. Since all particles in J_i and K_{ij} are included in M_i, each particle can interact with up to 320 particles. Hence, non-zero values for each velocity component can be 960, as there are 320 particles in M_i and each has three velocity components. Although this number is much larger than the number of non-zero values in

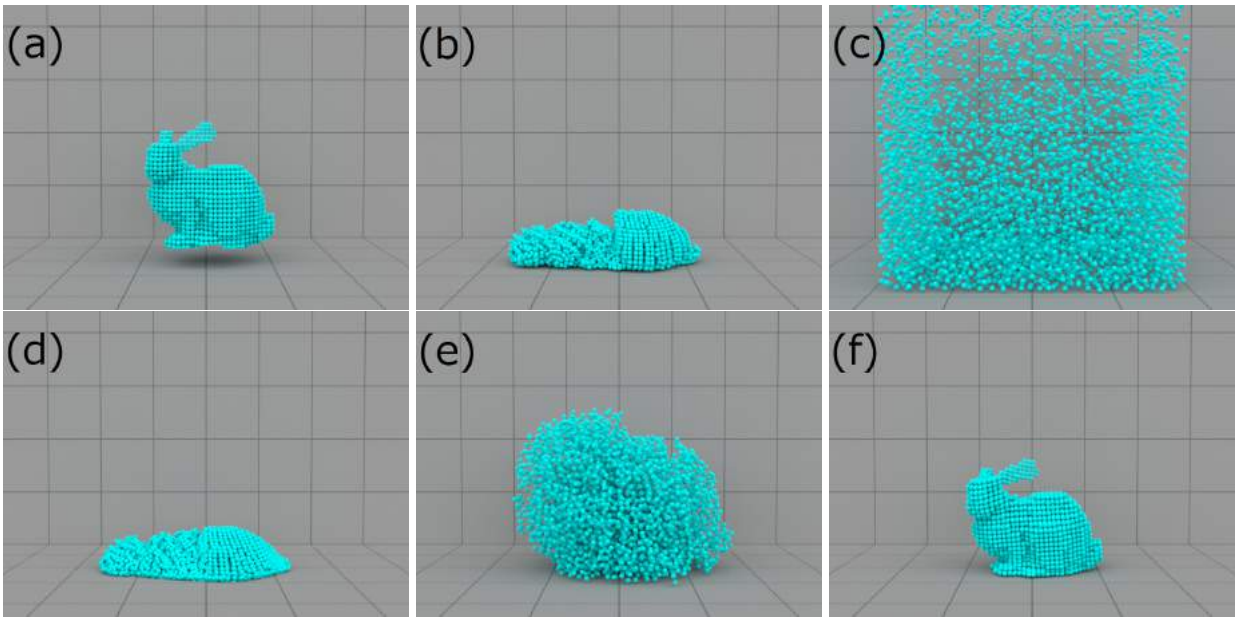


Figure 4.3: Numerical stability test with different combinations of time steps and viscosities. (a) initial state, (b) explicit integration (Andrade et al., 2014) with $\Delta t = 5.0 \times 10^{-6}$ s and $\mu = 1,000.0$ kg/(s·m), (c) explicit integration (Andrade et al., 2014) with $\Delta t = 1.3 \times 10^{-3}$ s and $\mu = 1,000.0$ kg/(s·m), (d) implicit integration (my method) with $\Delta t = 1.3 \times 10^{-3}$ s and $\mu = 1,000.0$ kg/(s·m), (e) explicit integration (Andrade et al., 2014) with $\Delta t = 5.0 \times 10^{-6}$ s and $\mu = 50,000.0$ kg/(s·m), and (f) implicit integration (my method) with $\Delta t = 1.0 \times 10^{-4}$ s and $\mu = 50,000.0$ kg/(s·m).

grid-based methods, and particle-based methods that involve only first-ring neighbors, my coefficient matrix is still a sparse matrix.

4.4.3 Solver and Coefficient Extraction

Solver. Since the linear system constructed by my formulation is sparse and also symmetric positive definite, I use a conjugate gradient (CG) solver (though I also tested Jacobi method and Modified Incomplete Cholesky Conjugate Gradient (MICCG), they did not work well. See § 4.6). Unlike the CG method described in (Ihmsen, 2013), which repeatedly computes matrix coefficients in the CG algorithm by performing extra particle scans without storing the coefficients, I explicitly construct and preserve a coefficient matrix, namely extract all coefficients in the matrix. This approach enables us to efficiently perform the CG method without extra loops, thereby improving the performance of the solver. In addition to this advantage, extracting all coefficients allows us to use preconditioning techniques, e.g., algebraic multigrid (AMG), and external libraries, separate solver code from others to improve programming maintainability, and take full advantage of GPGPU, parallelizing matrix-vector multiplications (in both rows and columns).

After a coefficient matrix is constructed, I solve the linear system with the CG method. I terminate iterations in the CG algorithm when a relative residual becomes smaller than a convergence criterion η.

Coefficient extraction. By substituting $\mathbf{Q}_{ij}$ in Eq. (4.8), I can rewrite Eq. (4.7) for x component of $\mathbf{u}_i$, u_i as

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(2\nabla W_{ij,x} \sum_j a_{ij,x} u_{ij} + \\
\nabla W_{ij,y} \sum_j (a_{ij,y} u_{ij} + a_{ij,x} v_{ij}) + \nabla W_{ij,z} \sum_j (a_{ij,z} u_{ij} + a_{ij,x} w_{ij}) \Big) \\
+ \hat{\mu}_j \Big(2\nabla W_{ij,x} \sum_k a_{jk,x} u_{jk} + \nabla W_{ij,y} \sum_k (a_{jk,y} u_{jk} + a_{jk,x} v_{jk}) \\
+ \nabla W_{ij,z} \sum_k (a_{jk,z} u_{jk} + a_{jk,x} w_{jk}) \Big) \Bigg) = u_i^*. \qquad (4.9)
\end{aligned}
$$

Then, I further convert Eq. (4.9) into the following equation to straightforwardly extract coefficients $c_{u_i u_i}, c_{v_i u_i}, c_{w_i u_i}, c_{u_j u_i}, c_{v_j u_i}, c_{w_j u_i}, c_{u_k u_i}, c_{v_k u_i}$, and $c_{w_k u_i}$:

$$
\begin{bmatrix} c_{u_i u_i} \\ c_{v_i u_i} \\ c_{w_i u_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_j u_i} \\ c_{v_j u_i} \\ c_{w_j u_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_k u_i} \\ c_{v_k u_i} \\ c_{w_k u_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = u_i^*,
$$

$$
\begin{aligned}
c_{u_iu_i} &= 1 + \hat{m}\hat{\mu}_i \left(2\omega_{ij,x}\alpha_{ij,x} + \omega_{ij,y}\alpha_{ij,y} + \omega_{ij,z}\alpha_{ij,z}\right), \\
c_{v_iu_i} &= \hat{m}\hat{\mu}_i\omega_{ij,y}\alpha_{ij,x}, \\
c_{w_iu_i} &= \hat{m}\hat{\mu}_i\omega_{ij,z}\alpha_{ij,x}, \\
c_{u_ju_i} &= \hat{m}\big(-\hat{\mu}_i(2a_{ij,x}\omega_{ij,x} + a_{ij,y}\omega_{ij,y} + a_{ij,z}\omega_{ij,z}) + \\
&\quad \hat{\mu}_j(2\nabla W_{ij,x}\alpha_{jk,x} + \nabla W_{ij,y}\alpha_{jk,y} + \nabla W_{ij,z}\alpha_{jk,z})\big), \\
c_{v_ju_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,y} + \hat{\mu}_j\nabla W_{ij,y}\alpha_{jk,x}\right), \\
c_{w_ju_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,z} + \hat{\mu}_j\nabla W_{ij,z}\alpha_{jk,x}\right), \\
c_{u_ku_i} &= -\hat{m}\sum_j \hat{\mu}_j(2\nabla W_{ij,x}a_{jk,x} + \nabla W_{ij,y}a_{jk,y} + \nabla W_{ij,z}a_{jk,z}),
\end{aligned}
\tag{4.10}
$$

$$
c_{v_ku_i} = -\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}a_{jk,x}, \tag{4.11}
$$

$$
c_{w_ku_i} = -\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}a_{jk,x}, \tag{4.12}
$$

where $\alpha_{ij} = [\alpha_{ij,x}, \alpha_{ij,y}, \alpha_{ij,z}]^T = \sum_j \mathbf{a}_{ij}$ and $\omega_{ij} = [\omega_{ij,x}, \omega_{ij,y}, \omega_{ij,z}]^T = \sum_j \nabla W_{ij}$. I use $c_{u_iu_i}$ to denote a coefficient of u_i to u_i, and $c_{v_iu_i}$ a coefficient of v_i to u_i, and similarly define other coefficients. The other components (y and z) and 2D version can be straightforwardly derived. By scanning particles i, j, and k, I extract all coefficients for the linear system (see Appendix B for details).

4.4.4 Implementation Details and Algorithm

I summarize procedures of my implicit formulation in Algorithm 4. To handle solid boundaries, I use solid particles arranged on object surfaces. When fluid particles collide with solid particles, I use explicit viscosity integration for fluid particles with low viscosity while using Dirichlet boundary condition similar to (Batty and Bridson, 2008; Stomakhin et al., 2014; Andrade et al., 2014), namely setting averaged solid particle velocities $\mathbf{u}^{\text{solid}}$ to fluid particles if viscosity of the fluid particles is higher than a criterion $\mu^{\text{Dirichlet}}$. As a structure of a sparse matrix, I use compressed sparse row (CSR) and reserve sufficient memory for each particle so that the matrix construction can be parallelized over particle i.

Algorithm 4 Algorithm for solving viscosity

```
assemble the matrix // see Appendix A
solve the linear system with CG
for all fluid particle i do
    if μ^Dirichlet < μ_i ∧ neighbor solid particle exists then
        enforce solid boundary condition u_i = u^solid
```

Table 4.1: Simulation parameters and performance. N: the number of particles, μ $(kg/(s \cdot m))$: dynamic viscosity of particles, Δt (s): time step, and t^{visc} (s) and t^{total} (s): simulation time for viscosity and total simulation time per frame, respectively.

Figure	Viscosity form	Integration	N	μ	Δt	t^{visc}	t^{total}
4.3 (b)	Full	Explicit	5.5k	1,000.0	5.0×10^{-6}	6.3	51.7
4.3 (c)	Full	Explicit	5.5k	1,000.0	1.3×10^{-3}	0.0	0.5
4.3 (d)	Full	Implicit	5.5k	1,000.0	1.3×10^{-3}	15.0	15.4
4.3 (e)	Full	Explicit	5.5k	50,000.0	5.0×10^{-6}	2.9	40.1
4.3 (f)	Full	Implicit	5.5k	50,000.0	1.0×10^{-4}	361.4	366.1
4.5	Full	Implicit	47.4k	up to 800.0	1.0×10^{-4}	815.2	836.1
4.6 (a), (c)	Laplacian	Implicit	up to 19.6k	600.0	2.0×10^{-4}	20.4	59.3
4.6 (b), (d)	Full	Implicit	up to 46.5k	600.0	2.0×10^{-4}	287.1	314.1
4.7 (a), (c)	Full	Implicit	up to 34.6k	100.0	2.0×10^{-4}	135.4	142.0
4.7 (b), (d)	Full	Implicit	up to 23.4k	600.0	2.0×10^{-4}	116.3	120.7

4.5 Results

Implementation. I implemented my method in C++ and parallelized it using Open MP 2.0. I adopted SPH kernels proposed in (Müller et al., 2003) and used surface tension force presented in (Becker and Teschner, 2007b). I adopted Implicit Incompressible SPH (IISPH) (Ihmsen et al., 2014a) as an incompressible fluid solver and used the fluid solid coupling method proposed in (Akinci et al., 2012). I used a variant of the z-index neighbor search method presented in (Ihmsen et al., 2011). In addition to my viscosity formulation, I also used artificial viscosity to stabilize simulations (Monaghan, 1992).

Setting. I executed all the scenes on a PC with a 4-core Intel Core i7 3.40 GHz CPU and RAM 16.0 GB, and rendered all the figures using a physically-based renderer, *Mitsuba*. I used fixed time steps and set a convergence criterion as $\eta = 1.0 \times 10^{-4}$. Simulation parameters and performance are listed in Table 4.1, where the surface reconstruction and rendering are not included in performance measurement.

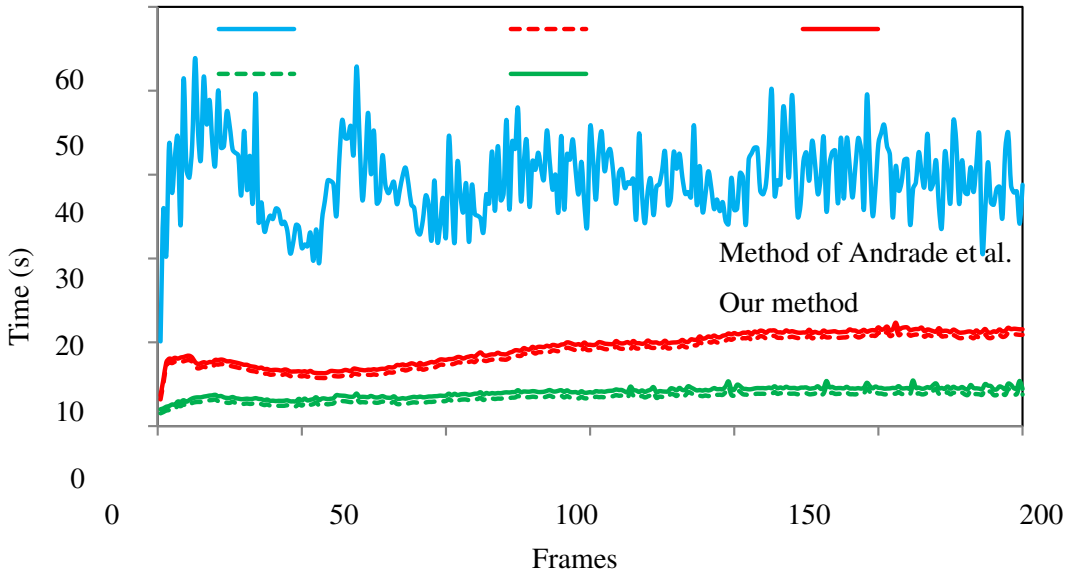


Figure 4.4: Performance profile for Figures 4.3 (b) and (d).

4.5.1 Numerical Stability

To verify the numerical stability of my implicit formulation over the previous method (Andrade et al., 2014) that uses explicit viscosity integration, I performed a simple test, as shown in Figure 4.3, where a viscous bunny was dropped onto the ground, and the initial state is shown in Figure 4.3 (a). In this scene, I chose Δt satisfying the viscosity condition ($\Delta t \leq 0.1\frac{\rho h^2}{8\mu}$) given in (Andrade et al., 2014), and used IISPH as a fluid solver of (Andrade et al., 2014) for fair comparison. The method of (Andrade et al., 2014) can generate a plausible behavior of the bunny with a small time step and low viscosity, as shown in Figure 4.3 (b). However, when a large time step or high viscosity is used, their method easily fails to simulate the bunny, as shown in Figures 4.3 (c) and (e). By contrast, my implicit method successfully simulates the bunny with a much larger time step (Figure 4.3 (d)), and with a large time step and high viscosity (Figure 4.3 (f)).

4.5.2 Performance

I compared performance of my method and the previous method (Andrade et al., 2014) in Figure 4.4 using the bunny scene shown in Figures 4.3 (b) and (d). Because of my robust implicit formulation, I can take a 260.0 times larger time step than the method of (Andrade et al., 2014), and my computation is 3.4 times faster than theirs although per step cost of my implicit method is more expensive than that of explicit methods. Since time steps for (Andrade et al., 2014) are restricted by the viscosity condition above, my method can be more advantageous for scenes with higher viscosities and finer resolutions.

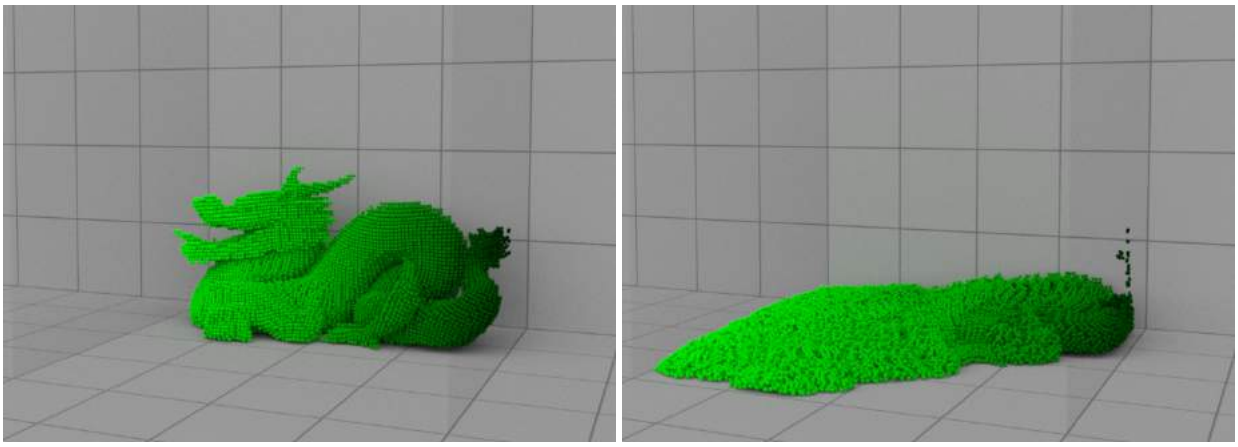


Figure 4.5: A dragon consisting of particles with different viscosities from 0.0 to 800.0 kg/(s·m). Light (dark) green particles represent low (high) viscosity.

4.5.3 Variable Viscosity

Figure 4.5 illustrates an example of a dragon consisting of particles with different viscosities from 0.0 (light green) to 800.0 kg/(m·s) (dark green). Particles flow at different rates depending on particle velocities.

4.5.4 Buckling and Coiling

I performed a buckling test to demonstrate that my implicit formulation can generate buckling of a viscous material while the Laplacian form with implicit integration fails to generate a buckling phenomenon. Figure 4.6 illustrates a dropped chocolate onto the ground, where (a) and (c) are simulated with the Laplacian form, and (b) and (d) are with my formulation. The chocolate simulated with the Laplacian form does not bend, and exhibits unnatural fluid flows regardless of its high viscosity. By contrast, my implicit formulation successfully generates natural chocolate buckling.

I also performed a coiling test with different viscosities in Figure 4.7. In this scene, I poured caramel sauce with low and high viscosity onto the ground. While caramel sauce with low viscosity ((a) and (c)) behaves similar to the results generated with the Laplacian form in Figures 4.6 (a) and (c), caramel sauce with high viscosity exhibits a coiling phenomenon ((b) and (d)).

4.6 Discussions and Limitations

Robustness. My implicit formulation significantly improves the robustness of viscosity integration and allows us to avoid the time step restriction for explicit integration (Andrade et al., 2014). My method

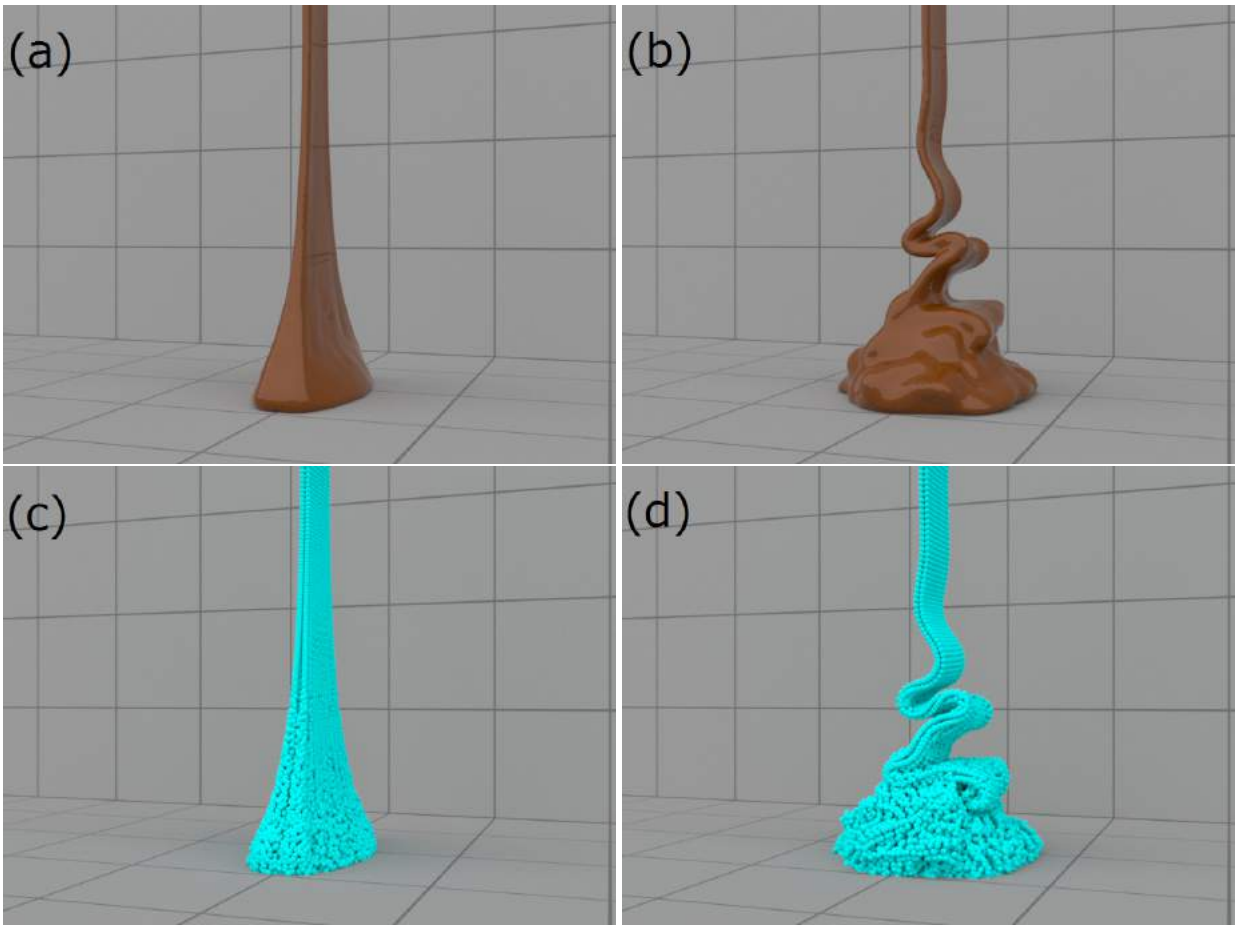


Figure 4.6: Buckling test for comparison of my method with the Laplacian form using implicit integration. (a) Laplacian form with meshes. (b) my method with meshes. (c) Laplacian form with particles. (d) my method with particles.

introduces a much more relaxed restriction on time steps, which depends only slightly on viscosity and spatial resolutions. Consequently, my method may not generate plausible fluid behaviors, when very large time steps and very high viscosity and resolutions are used, even though an implicit formulation for the Laplacian form (which is unconditionally stable) can perform stable simulations with the same condition.

From my experiments, I deduce that one decisive factor for this weaker robustness is due to second-ring neighbors. To examine the factor, I consider solving differential equations with Laplacian operator by using SPH formulations for Laplacian, and also solving the equations by separating Laplacian into divergence and gradient, applying SPH formulations for both of divergence and gradient operators, and including second-ring neighbors, because $\beta\nabla^2\phi \approx \nabla\cdot(\beta\nabla\phi)$ (β and ϕ are arbitrary quantities), and this decomposition is numerically similar to the relation of the Laplacian form $\mu\nabla^2\mathbf{u}$ and full form $\nabla\cdot(\mu\nabla\mathbf{u}+\mu(\nabla\mathbf{u})^T)$. I actually tested these with the Laplacian form of viscosity and heat equations, and as expected, I observed

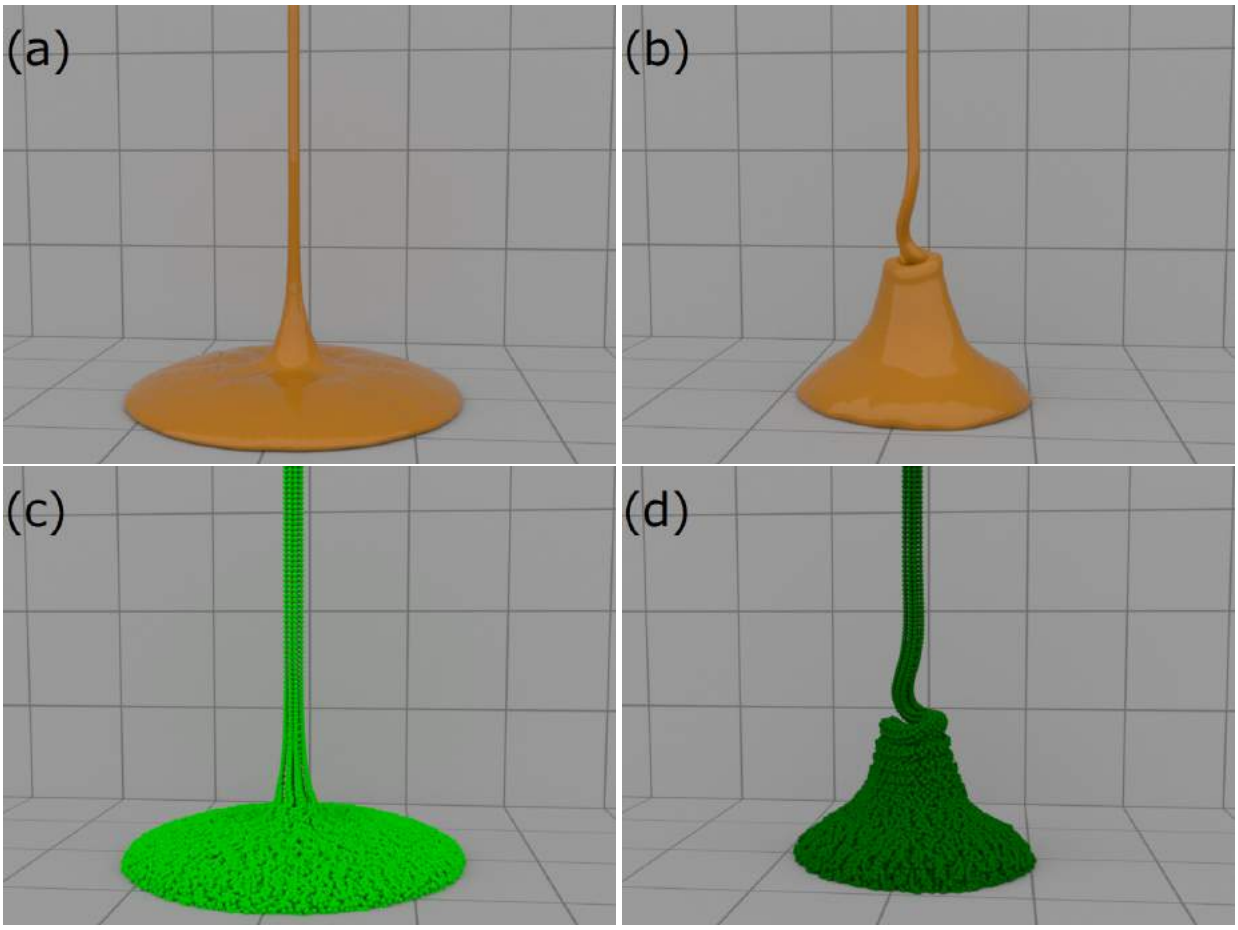


Figure 4.7: Coiling test for different viscosities. (a) Low viscosity with meshes. (b) High viscosity with meshes. (c) Low viscosity with particles. (d) High viscosity with particles. Light (dark) green particles represent low (high) viscosity.

numerical instability with larger time steps and higher coefficients and resolutions when Laplacian operator was decomposed while I was able to perform stable simulations with undecomposed Laplacian operator under the same condition.

Solver. In addition to CG, I tested Jacobi method and MICCG to solve my linear system. Although I was able to solve the system using the two solvers, there are a few problems to note. First, Jacobi method worked for slightly viscous fluids with smaller time steps under low spatial resolutions. However, since the convergence rate of Jacobi method is slow, Jacobi method failed to solve the system with large time steps and high viscosity and spatial resolutions that can make a coefficient matrix ill-conditioned. In contrast to Jacobi method, MICCG worked for such an ill-conditioned matrix, and the rate of convergence was actually faster than my CG solver. However, MICCG requires Cholesky factorization that is costly (and difficult to parallelize), especially with my matrix that includes a great number of non-zero values (see § 4.4) although

serial forward and back substitution steps are not so slow compared to other steps in the CG algorithm. In my experiments, Cholesky factorization occupied more than 90% of computational time for my viscosity solver including coefficient extractions, and therefore my non-preconditioned CG solver was more than 6 times faster than MICCG. A similar behavior that the incomplete Cholesky preconditioner performed worse due to the cost of constructing a preconditioning matrix for particle-based methods (many non-zeros) was also reported in (He et al., 2012b).

Performance. Solving my viscosity formulation generally occupies more than 90% of the whole computational time, and this weakens one of my advantages of efficient computation with larger time steps. My viscosity formulation consists of extracting coefficients through triple loops and solving a linear system, and they occupy around 30% and 70% of computational time, respectively, for Figure 4.3 (f). To accelerate the speed of coefficient extractions, an efficient algorithm for extracting coefficients involving second-ring neighbors would be useful. One possible fast algorithm is to avoid triple loops when extracting coefficients, and I can avoid them by using precomputations for particle k. However, these computations require additional storage and scans over the storage, making coefficient extractions more complex. Consequently, benefits of the precomputations over my method can be lost, or extracting coefficients with the precomputations can be more costly than mine. As for solving a linear system, using a faster solver or a low cost and effective preconditioner would be helpful.

Memory. Preserving a coefficient matrix requires a large memory (e.g., 12 GB memory for 500k particles, due to 1k of 8 byte double values for 3 velocity components of 500k particles). However, this is not a big issue with current memory capacity, given advantages explained in § 4.4.3.

Scalability. Another issue to note is that the size of a matrix grows proportionally to the number of particles. It has a scaling factor larger than grid-based methods and particle-based methods that involve only first-ring neighbors. On larger scenes with up to 200k particles, the memory usage and computational cost for coefficient extraction increased at nearly 1.0x linear-scale with respect to the number of particles – similar to grid-based methods and particle-based methods involving only first-ring neighbors – whereas computational cost for solving linear system increased at around 1.1x scale. This is because increased number of particles requires more CG iterations and thus more computational time. Although computational cost might increase superlinearly when more particles are used, using AMG preconditioners (which cannot be used without explicit matrix preservation) can potentially address this problem.

4.7 Conclusion and Future Work

I proposed a new SPH-based implicit formulation for the full form of viscosity. My method enables efficient and stable viscous fluid simulations with larger time steps and higher viscosities and resolutions than previous methods that use explicit integration while handling variable viscosity and generating coiling and buckling. I additionally presented a novel coefficient extraction method for a sparse matrix that involves second-ring neighbors to efficiently solve a linear system with a CG solver. By taking advantage of my implicit formulation and coefficient extraction method, I achieved an accelerated performance by a factor of 3.4.

For future work, I plan to implement my method using GPGPU techniques to accelerate constructing and solving the linear system. In particular, my explicit matrix preservation allows us to take full advantage of dynamic parallelism, fully parallelizing matrix-vector multiplications. Additionally, finding better solvers, efficient and effective preconditioners as well as fast coefficient extraction methods would be promising. Similar to improving memory and computational efficiency for enforcing incompressibility in SPH fluids, using multi-sized particles, domain decomposition, and background grids could also further optimize performance.

My coefficient extraction method can be applied to not only SPH, but also other point-based methods that involve second-ring neighbors (IISPH (Ihmsen et al., 2014a) and mesh smoothing (Desbrun et al., 1999)). Extending my method to accelerate such problems could also lead to interesting future research directions.

Acknowledgements

This work is supported in part by JASSO for Study Abroad, JST CREST, JSPS KAKENHI (Grant-in-Aid for Scientific Research(A)26240015), U.S. National Science Foundation, and UNC Arts and Sciences Foundation. I would like to thank Ryoichi Ando and anonymous reviewers for their valuable suggestions and comments.

CHAPTER 5: A Geometrically Consistent Viscous Fluid Solver with Two-Way Fluid-Solid Coupling

5.1 Introduction

Viscous fluids, such as honey, molten chocolate, paint, oil, and shampoo, are common materials as frequently seen in daily life. These viscous materials exhibit characteristic behaviors including damped and sticky motions and buckling phenomena, both of which are not observed for inviscid fluids. Because of the ubiquity of viscous fluids and their fascinating behaviors, simulating viscous fluids has been needed in a variety of applications e.g., video games, feature films, and virtual reality. Various researchers have proposed simulation methods specifically designed for viscous fluids (Carlson et al., 2002; Rasmussen et al., 2004; Batty and Bridson, 2008; Bergou et al., 2010; Batty et al., 2012; Zhu et al., 2015; Takahashi et al., 2015; Peer et al., 2015; Larionov et al., 2017).

In the previous works, several compelling viscous fluid behaviors have been demonstrated focusing primarily on simulating the intriguing behaviors of viscous fluids induced by the gravitational forces with static or prescribed solid boundaries. However, mutual interactions between viscous fluids and solid objects are essential, and thus it is necessary to correctly handle such interactions. In the literature, there are various approaches presented to simulate the interactions of *inviscid* fluids and solids objects (Carlson et al., 2004; Klingner et al., 2006; Chentanez et al., 2006; Batty et al., 2007; Robinson-Mosher et al., 2008; Lu et al., 2016; Teng et al., 2016; Zarifi and Batty, 2017). These approaches enable the two-way interactions between fluids and solid objects with pressure forces, which play roles of drag and buoyancy forces. The pressure forces can be sufficient to describe the interactions between nearly inviscid fluids and solid objects. However, simulating the interactions between highly viscous fluids and solid objects with only pressure forces leads to fluid and solid behaviors significantly different from those observed in the real world, and it is necessary to consider *viscosity* forces to correctly account for the two-way interactions.

In this chapter, I present a grid-based fluid solver that can handle two-way interactions of fluids and solid objects with viscosity forces. My method formulates the dynamics of viscous fluids and solid objects as a unified minimization problem based on the variational principle, and thus achieves the correct behaviors for

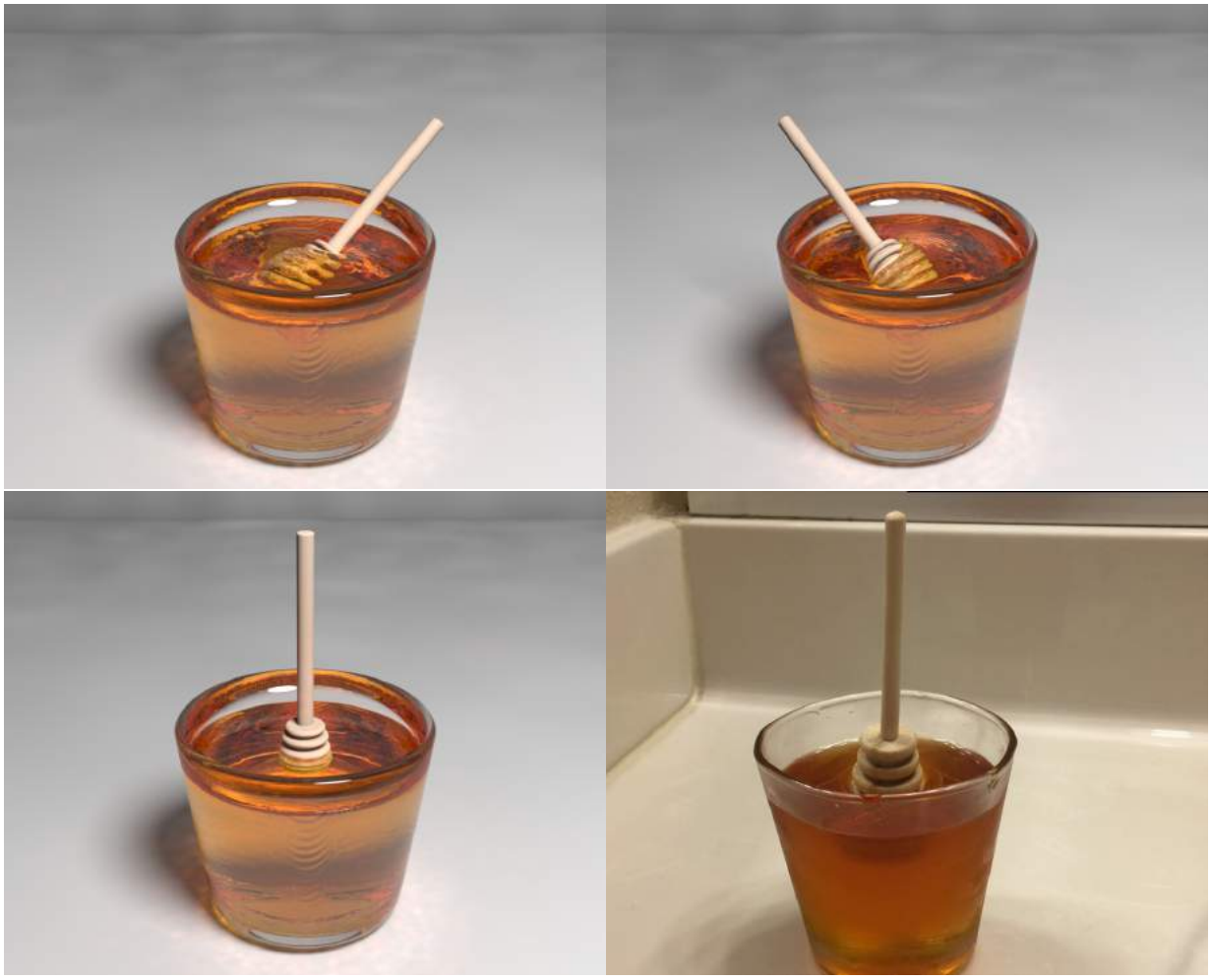


Figure 5.1: A wooden honey dipper in the honey. (Left to right): simulated honey with one-way coupling, weak two-way coupling, strong two-way coupling, and the real honey. While the simulated honey using one-way and weak two-way coupling cannot sufficiently support the honey dipper, the simulated honey using my strong two-way coupling can keep the honey dipper standing, as observed in the real phenomena.

both fluids and solids. To improve the accuracy even on relatively coarse grids, I estimate volume fractions of free surfaces and solid boundaries in a geometrically consistent way and formulate my method with these fractions accounting for the sub-grid level details. In my method, I use Lagrangian particles to discretize the fluid volumes and address the advection by moving particles. While the combination of particles and a grid is proven to be effective (Bridson, 2015), one known issue is that the distributions of particles would be non-uniform leading to the loss of fluid volumes and producing gaps and holes in the fluids. In addition, the interactions with solids can make the particle distributions even more non-uniform further causing undesirable volume changes. To address this issue, I propose a position-correction method based on density constraints at

the particle level to enforce the uniform particle distributions. In summary, my method offers the following key contributions:

- **A geometrically consistent volume fraction estimation** that utilizes the supersampling technique to improve the accuracy of the simulation and avoids dangling artifacts near free surfaces and solid boundaries (§ 5.3.3).
- **A two-way fluid-solid coupling using viscosity force** that robustly and correctly accounts for the dynamics of viscous fluids and solid objects through their interactions (§ 5.3.4).
- **A position-correction method** using density constraints with a position-correction scaling that enforces the uniform particle distributions avoiding non-physical volume changes (§ 5.3.5).

I integrate these techniques with a viscous fluid solver. Figure 5.1 demonstrates complex interactions of viscous fluids and solid objects simulated with my method.

5.2 Related Work

In this section, I discuss previous works closely related to mine. I refer to (Bridson, 2015) for the literature and fundamentals.

5.2.1 Viscous Fluids

In the Eulerian framework, an early work, stable fluid method (Stam, 1999) solved the Navier-Stokes equations with implicit viscosity integration for numerical stability while focusing on fluids without free surfaces. Carlson et al. (Carlson et al., 2002) proposed the first method for simulating highly viscous fluids with free surfaces by using a simplified, Laplacian-based viscosity model with implicit integration. However, the Laplacian-based formulation immediately damps the rotational velocity fields due to incorrect free surface boundary conditions. Later, Rasmussen et al. (Rasmussen et al., 2004) augmented the Laplacian-based formulation by adding off-diagonal components with explicit integration while sacrificing the robustness. Batty and Bridson (Batty and Bridson, 2008) proposed a fully implicit integration scheme for the full form of viscosity with correct free surface boundary conditions and made it possible to simultaneously take larger time steps, handle variable viscosity, and generate rotational fluid motions. Later, this method was extended for an adaptive tetrahedral fluid simulator (Batty and Houston, 2011). Recently, Larionov et al. (Larionov

et al., 2017) proposed a pressure-viscosity coupled solver to further improve the accuracy in the free surface handling. While Robinson-Mosher et al. (Robinson-Mosher et al., 2011) also presented a pressure-viscosity coupled solver, their approach focused on fluids without free surfaces adopting the Laplacian-based viscosity formulation with the voxel-based discretization. Although the previous methods (Batty and Bridson, 2008; Larionov et al., 2017) used volume fractions for the sub-grid level accuracy, they did not address how to consistently estimate control volumes of fluids and solids for velocity and viscous stress.

To simulate more general fluids, e.g., viscoelastic fluids and non-Newtonian fluids, various approaches were also proposed. Goktekin et al. (Goktekin et al., 2004) presented a method for simulating viscoelastic fluids by adding an extra term for elastic forces. Recently, to handle fluid-like materials with a variety of properties in a unified way, material point methods (MPM) have been widely adopted with some specialized extensions for snow (Stomakhin et al., 2013), foams (Yue et al., 2015), melting solids (Stomakhin et al., 2014), and granular materials (Daviet and Bertails-Descoubes, 2016; Klár et al., 2016; Jiang et al., 2017; Tampubolon et al., 2017; Gao et al., 2018; Hu et al., 2018). While these approaches allow us to simulate a wide range of materials, the constitutive laws adopted in these works typically involve the non-linearity which would cause stability issues with explicit integration or requires expensive non-linear solves for implicit integration. Thus, in this chapter, I focus on purely Newtonian viscous fluids, whose dynamics on viscosity can be simulated with only linear solves.

In the Lagrangian framework, various approaches have been also proposed to simulate viscous fluids. Specifically, particle-based methods based on Smoothed Particle Hydrodynamics (SPH) have been widely used, and recently, various extensions were presented to improve the robustness and efficiency of the viscosity integration (Takahashi et al., 2015; Peer et al., 2015; Bender and Koschier, 2016; Peer and Teschner, 2017; Barreiro et al., 2017; Weiler et al., 2018a). In the Lagrangian setting, some researchers proposed dimension-reduced representations to capture the detailed dynamics of viscous threads and sheets (Bergou et al., 2010; Batty et al., 2012; Zhu et al., 2015). While my method employs Lagrangian particles for advection, surface tracking, and volume preservation, the dynamics is computed on a grid unlike these purely Lagrangian approaches.

5.2.2 Two-Way Fluid-Solid Coupling

For coupling of Eulerian fluids with Lagrangian solids, early works, e.g., (Guendelman et al., 2005) achieved the two-way coupling with pressure forces by alternatively solving one-way fluid-to-solid and

solid-to-fluid coupling, which is known as weak two-way coupling. While the weak coupling would work in certain scenarios, the stability of the simulation is not guaranteed, and it is generally necessary to take very small time steps with many alternate one-way solves. Carson et al. (Carlson et al., 2004) presented a two-way coupling method that temporarily treats solid objects as fluids in the pressure solve sacrificing the robustness. Chentanez et al. (Chentanez et al., 2006) presented a two-way coupling method that simultaneously considers both dynamics of fluids and deformable solids, which is known as strong two-way coupling. While their method improves the stability, the resulting linear system is not symmetric. Klingner et al. (Klingner et al., 2006) also proposed a two-way coupling approach with dynamic meshes. While their linear system is symmetric positive definite (SPD), the computational cost is likely to be expensive due to the dynamic meshes. Batty et al. (Batty et al., 2007) presented a strong two-way coupling method with the Cartesian grid based on the variational principle and significantly improved the efficiency. Later, this method was extended for frictional forces (Narain et al., 2010). Robinson-Mosher et al. (Robinson-Mosher et al., 2008) also presented a two-way coupling approach with a more accurate momentum handling than (Batty et al., 2007). Later, they extended their approach to improve the accuracy of tangential velocities (Robinson-Mosher et al., 2009) and to generate an SPD system with the Laplacian form of viscosity (Robinson-Mosher et al., 2011), and the performance of this approach was improved with a multigrid preconditioning (Aanjaneya, 2018). Recently, strong two-way coupling was employed for interactions between deformable solids and fluids (Lu et al., 2016; Teng et al., 2016; Zarifi and Batty, 2017), interactions between rigid bodies and fluids simulated using a vorticity-based fluid solver (Vines et al., 2014) and a stream function solver (Ando et al., 2015), while weak two-way coupling was augmented with the reduced model interface to improve the stability (Akbay et al., 2018).

In the MPM framework, collisions between fluids and other objects can be naturally handled at the grid level. Because of this advantage, MPM is widely adopted to simulate two-way interactions with rigid bodies (Daviet and Bertails-Descoubes, 2016; Hu et al., 2018) and with deformable solids (Klár et al., 2016; Jiang et al., 2017). Since solid objects are typically described in a Lagrangian setting, Lagrangian particle-based methods are also extensively used, and the two-way coupling can be achieved in a unified way (Solenthaler et al., 2007; Akinci et al., 2012; Macklin et al., 2014).

While there are various two-way coupling methods proposed for Eulerian fluids and Lagrangian solids, most of these approaches focus on two-way coupling with pressure forces, but not viscosity forces. Although Robinson-Mosher et al. (Robinson-Mosher et al., 2011) presented a two-way coupling method with viscosity

force, their formulation relies on the Laplacian form of viscosity (which precludes rotational behaviors of viscous fluids) and focuses on voxel-based discretization with no free surfaces. In contrast to these approaches, my method focuses on two-way coupling with *viscosity* forces formulated using the *full* form of viscosity with free surfaces taking volume fractions into account for sub-grid details.

5.3 My Method

In this section, I describe my fluid solver using the implicit viscosity integration. I formulate the viscosity integration as a minimization problem based on the variational principle (§ 5.3.1) so that the minimization problem can be naturally discretized with volume fractions to account for the sub-grid level details (§ 5.3.2). To avoid dangling artifacts, I describe my geometrically consistent volume estimation (§5.3.3). Next, I present a two-way coupling formulation, which can be efficiently solved with the minimization problem in a unified way (§ 5.3.4). Then, I describe a position-correction method using density constraints to enforce the uniform distributions of particles (§ 5.3.5). Finally, I discuss previously proposed methods vs. mine to clarify key differences (§ 5.3.6).

5.3.1 Implicit Viscosity Formulation

The incompressible Navier-Stokes equations are given by

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla\cdot\mathbf{s} + \frac{1}{\rho}\mathbf{f}, \quad \mathbf{s} = \eta\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right),$$
$$\nabla\cdot\mathbf{u} = 0,$$

where t denotes time, $\frac{D}{Dt}$ material derivative, $\mathbf{u}$ velocity, ρ density, p pressure, $\mathbf{s}$ symmetric viscous stress tensor, $\mathbf{f}$ external force, and η dynamic viscosity. I address the advection term with the affine particle-in-cell (APIC) approach (Jiang et al., 2015) and take the operator splitting method to handle external force, pressure, and viscosity terms, applying solid boundary condition $\mathbf{u}^{t+1} = \mathbf{u}^{\text{solid}}$ ($\mathbf{u}^{\text{solid}}$: solid boundary velocity) and free surface boundary condition $\mathbf{sn} = 0$ ($\mathbf{n}$: outward unit normal of the free surface) for the viscosity solve (Batty and Bridson, 2008).

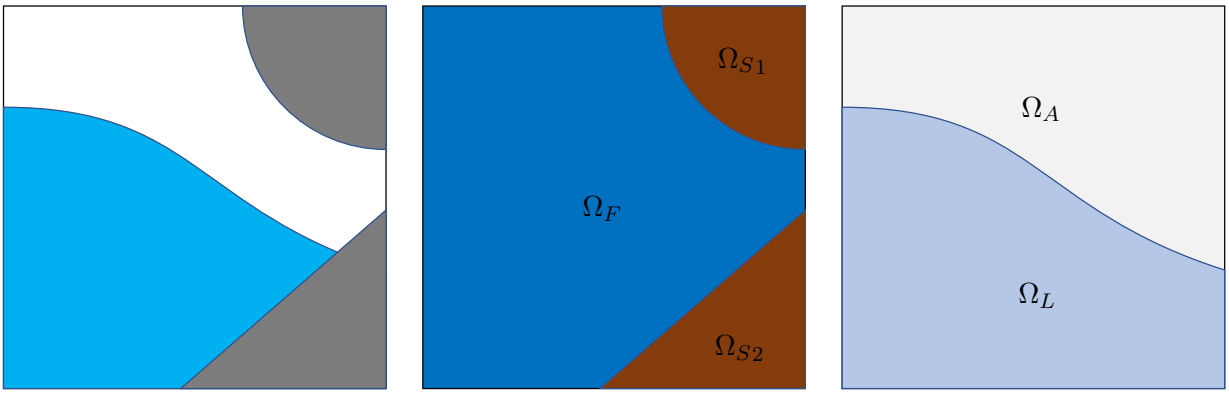


Figure 5.2: Domain illustration. (Left) The simulation domain Ω is filled with viscous fluids (cyan), solid objects (gray), and the rest (white). (Middle) The simulation domain is separated by the solid boundaries into multiple solid (brown) and fluid (blue) domains. (Right) The simulation domain is separated by the free surfaces into liquid (light blue) and air (light gray) domains.

The viscosity update with backward Euler can be written as

$$\frac{\mathbf{u}^{t+1} - \mathbf{u}^*}{\Delta t} = \frac{1}{\rho}\nabla \cdot \mathbf{s}^{t+1}, \quad \mathbf{s}^{t+1} = \eta\left(\nabla \mathbf{u}^{t+1} + \left(\nabla \mathbf{u}^{t+1}\right)^T\right),$$

where $\mathbf{u}^*$ denotes intermediate velocity after advection, external force, and first pressure projection steps (Batty and Bridson, 2008), and Δt time step size. In the following, I omit superscript $t+1$ for readability. These formulations can be cast as a minimization problem due to the variational principle (Batty et al., 2007; Batty and Bridson, 2008; Larionov et al., 2017):

$$\mathbf{s} = \arg\min_{\mathbf{s}} \int_{\Omega_F} \left(\frac{\rho}{2}\|\mathbf{u}^* + \frac{\Delta t}{\rho}\nabla \cdot \mathbf{s}\|^2 + \frac{\Delta t}{4\eta}\|\mathbf{s}\|_F^2\right) d\Omega, \tag{5.1}$$

where Ω_F denotes the fluid domain (see Figure 5.2), $\|\cdot\|_F$ the Frobenius norm. When dynamic viscosity η approaches to 0, viscous stress $\mathbf{s}$ (which includes η) also approaches to $\mathbf{0}$, and this minimization preserves intermediate velocity $\mathbf{u}^*$ (i.e., no viscosity force applied). On the other hand, when η approaches to ∞, optimal viscous stress are sought by minimizing the sum of these two terms. This will eventually prioritize the first term due to $\mathbf{s}$'s quadratic property with respect to η and make the term 0, giving certain viscous stress which leads to $\mathbf{u}\ (= \mathbf{u}^* + \frac{\Delta t}{\rho}\nabla \cdot \mathbf{s}) = \mathbf{0}$.

5.3.2 Discretization

To discretize Eq. (5.1), I approximate the integral in the minimization with fractions of cell-sized control volumes. Given solid boundaries defined with signed distance functions (SDF), simulation domain Ω is divided into multiple solid domains $\Omega_{S1}, \Omega_{S2}, \ldots, \Omega_{Sn}$ (n: number of solid objects) and fluid domains Ω_F, (i.e., $\cup_i^n \Omega_{Si} \cup \Omega_F = \Omega$, $\Omega_{Si} \cap \Omega_{Sj} = \phi$, and $\Omega_{Si} \cap \Omega_F = \phi$, where i, j $(i \neq j)$ denote an index for solid domains), as illustrated in Figure 5.2. Following the notations in (Larionov et al., 2017), I denote volume fractions for solid and fluid domains as diagonal matrices $\mathbf{W}_S^u$ and $\mathbf{W}_F^u$, respectively. I note that $\mathbf{W}_S^u$ is defined for each solid object, i.e., $\mathbf{W}_{S1}^u, \ldots, \mathbf{W}_{Sn}^u$ and $\sum_i \mathbf{W}_{Si}^u + \mathbf{W}_F^u = \mathbf{I}$. As for viscous stress defined in a staggered manner (Goktekin et al., 2004), I also compute volume fractions for solid domains $\mathbf{W}_S^s$ and fluid domains $\mathbf{W}_F^s$.

Similar to (Larionov et al., 2017), I can formulate a minimization problem for liquid domains (Figure 5.2), corresponding to Eq. (5.1). For discretization, as done for solid/fluid volume fractions, I can consider air/liquid volume fractions. Since free surfaces separate the simulation domain Ω into liquid domains Ω_L and air domains Ω_A, I also compute volume fractions for velocity components of liquid domains $\mathbf{W}_L^u$ and air domains $\mathbf{W}_A^u$ ($\mathbf{W}_L^u + \mathbf{W}_A^u = \mathbf{I}$), and viscous stress components of liquid and air domains as $\mathbf{W}_L^s$ and $\mathbf{W}_A^s$, respectively.

Combining the minimization formulations for fluid domains and liquid domains (Larionov et al., 2017), I obtain the following discretized minimization problem:

$$\mathbf{s} = \arg\min_{\mathbf{s}} \Big(\frac{1}{2} \| (\mathbf{P}\mathbf{W}_F^u \mathbf{W}_L^u)^{\frac{1}{2}} (\mathbf{u}^* - \Delta t \mathbf{P}^{-1} \mathbf{W}_L^{u\,-1} \mathbf{D}^T \mathbf{W}_L^s \mathbf{s} \|^2 + \frac{\Delta t}{4} \mathbf{N}^{-1} \| (\mathbf{W}_F^s \mathbf{W}_L^s)^{\frac{1}{2}} \mathbf{s} \|^2 \Big), \tag{5.2}$$

where $\mathbf{P}$ denotes a diagonal density matrix, $\mathbf{D}$ a discrete finite-difference operator, $\mathbf{N}$ a diagonal dynamic viscosity matrix.

5.3.3 Geometrically Consistent Volume Estimation

While I can compute volume fractions $\mathbf{W}_F^u, \mathbf{W}_L^u, \mathbf{W}_F^s$, and $\mathbf{W}_L^s$ for each cell *independently*, I found that evaluating these volume fractions in this way causes some artifacts near solid boundaries and free surfaces (e.g., dangling fluid particles in the air) due to the inconsistency of the estimated volumes over the

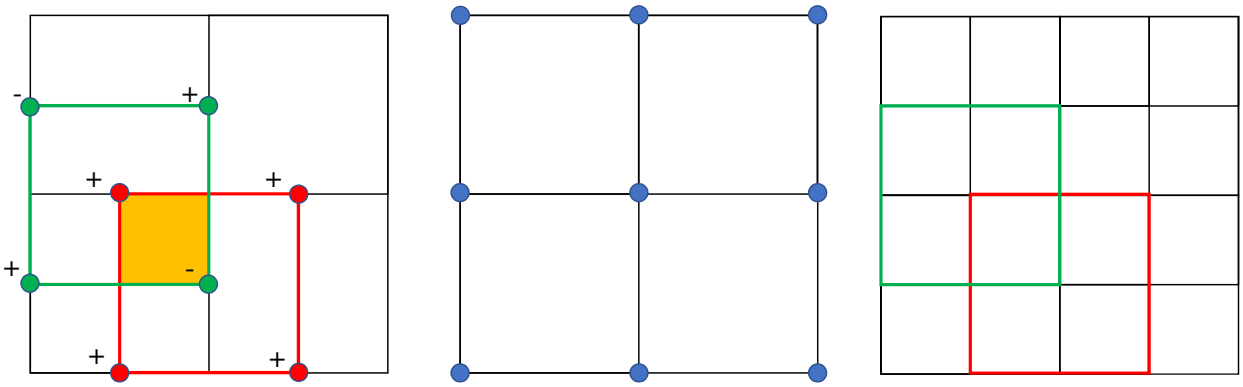


Figure 5.3: (Left) Illustration for inconsistent volumes with the independent volume estimation. Red and green squares represent u- and v-cells, respectively, and orange filled square represents an overlapping region between u- and v-cells. Red and green dots represent cell nodes, where SDF are evaluated for the volume computation of u- and v- cells, respectively. + and - represent signs of SDF at the node positions. (Middle) Simulation grid, where SDF for both solid boundaries and free surfaces are defined on the grid nodes (blue dots). (Right) Double-resolution grid, where volumes of each cell consistently contribute to the volume summation of u- (red) and v- (green) cells on the simulation grid.

domain. For example, as illustrated in Figure 5.3 (left), since volumes for u-cell (red) and v-cell (green) are independently evaluated with SDF at their cell nodes (red and green dots for u- and v-cells, respectively), it would happen that the u-cell has zero volumes (all signs of SDF are plus) while the v-cell has non-zero volumes leading to the volume inconsistency at the overlapping region (orange filled square).

To avoid this issue, I evaluate volume fractions in a geometrically consistent manner. I first construct a grid with a doubled resolution compared to the simulation grid, and evaluate volume fractions of the cells on the double-resolution grid with SDF defined on each node of the simulation grid (see Figure 5.3). Next, I sum up these volume fractions computed on the double-resolution grid to account for one volume fraction for the simulation grid, i.e., in 2D, I sum up four volume fractions on the double-resolution grid to account for u- and v- cells emphasized by red and green squares, respectively, in Figure 5.3 (right). Since volume fractions of each cell on the double-resolution grid consistently contribute to the volume summation on the simulation grid, this approach enforces the volume consistency over the domain.

The volume fractions for viscous stress can be similarly estimated by summing up the fractions computed on the double-resolution grid, and these computations can be naturally extended into 3D. In 2D, I evaluate the volume fractions using a marching-squares-style area computation method. In 3D, I use a volume

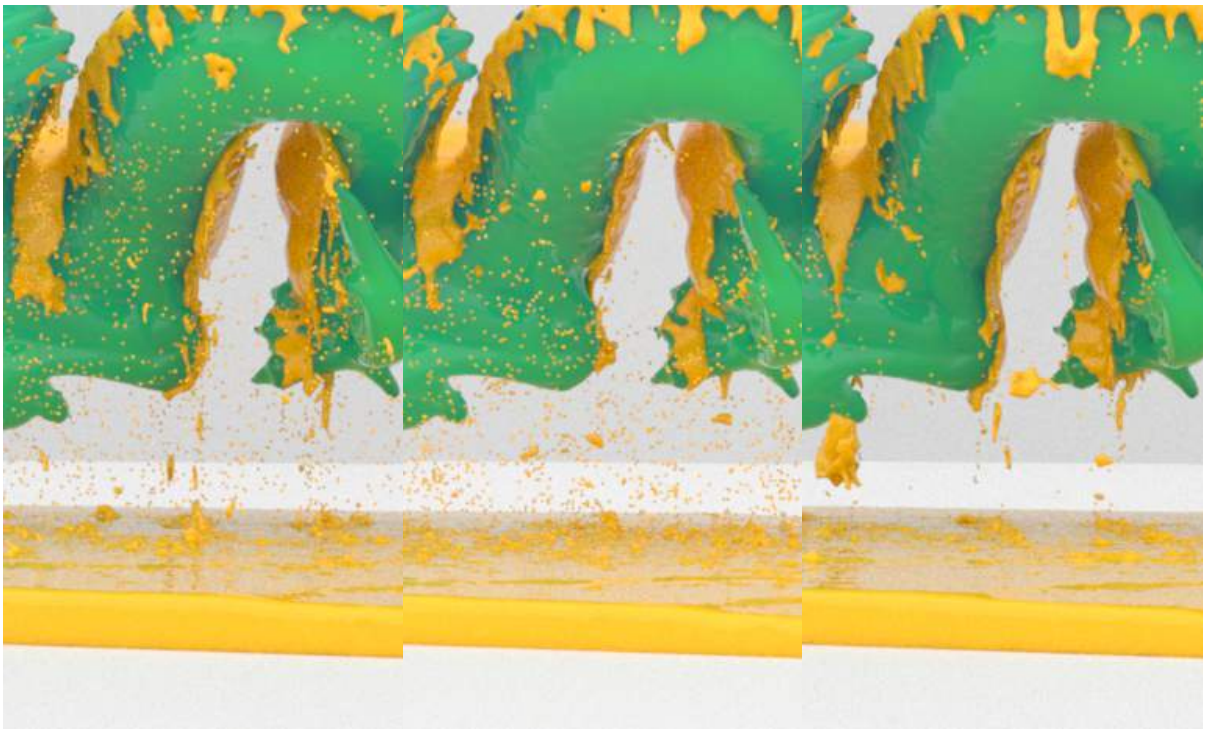


Figure 5.4: A viscous ball dropped onto a static, tilted solid dragon. The independent volume estimation causes artifacts that particles unnaturally float under the dragon due to the inconsistent volumes (left), and the supersampling method similarly suffers from the artifacts due to the inaccurate volume estimation (middle), whereas my geometrically consistent volume estimation method does not have such issues (right).

computation algorithm using the divergence theorem (Wang, 2013), which was around five times faster than the volume computation based on the tetrahedral decomposition (Min and Gibou, 2007).

My volume estimation approach can be considered as a generalized version of the volume estimation using the supersampling method that determines the fluid volumes based on level-set values at specific points(Bridson, 2015), and I found that a similar consistent volume computation method was recently used in (Batty, 2008) employing the volume computation based on (Min and Gibou, 2007), unlike my method based on (Wang, 2013).

5.3.4 Strong Two-Way Fluid-Solid Coupling

Similar to (Narain et al., 2010), I can describe a rigid body update due to the viscous stress applied from fluids with volume fractions by

$$\mathbf{V}^{t+1} = \mathbf{V}^t + \Delta t \mathbf{M}^{-1} \mathbf{W}_S^u \mathbf{J} \mathbf{W}_L^s \mathbf{s}, \tag{5.3}$$

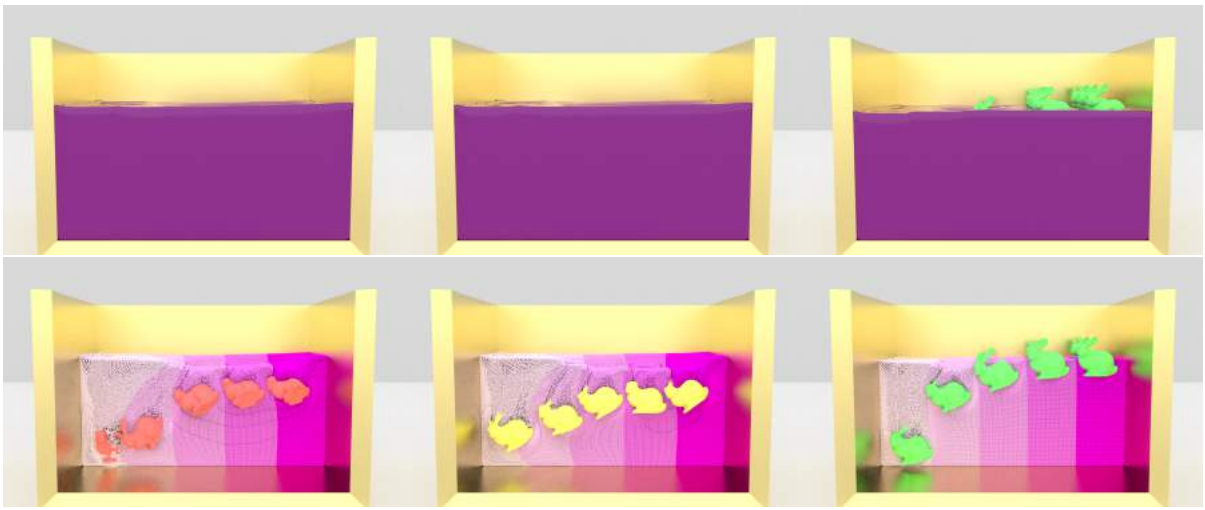


Figure 5.5: Multiple solid bunnies dropped onto fluids with spatially different viscosity values $\eta = 1.0 \times 10^1, 1.0 \times 10^2, 1.0 \times 10^3, 1.0 \times 10^4$, and 1.0×10^8 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. From left to right, one-way coupling, weak two-way coupling, and my strong two-way coupling, for rendered fluid surfaces (Top) and particle view, color-coded based on viscosity values (Bottom), where white and purple represent low and high viscosity values, respectively. Fluids simulated with one-way coupling and weak two-way coupling do not sufficiently reflect different viscosity values, whereas fluid simulated with strong two-way coupling correctly does.

where $\mathbf{V} \in \mathbb{R}^6$ denotes a generalized rigid body velocity, $\mathbf{M}$ a diagonal, generalized rigid body mass matrix, and $\mathbf{J}$ a linear operator which integrates viscous stresses over the surface of the rigid body to give viscosity forces. The rigid body update can be also cast as a minimization problem (Batty et al., 2007; Narain et al., 2010), and due to the minimization problem for fluids (Eq. (5.2)), I can naturally integrate these problems into the following, which monolithically couples viscous fluid and rigid body dynamics:

$$\mathbf{s} = \arg\min_{\mathbf{s}} \Big(\frac{1}{2} \|(\mathbf{P}\mathbf{W}_F^u \mathbf{W}_L^u)^{\frac{1}{2}} (\mathbf{u}^* - \Delta t \mathbf{P}^{-1} \mathbf{W}_L^{u\,-1} \mathbf{D}^T \mathbf{W}_L^s \mathbf{s}\|^2 + \frac{\Delta t}{4} \mathbf{N}^{-1} \|(\mathbf{W}_F^s \mathbf{W}_L^s)^{\frac{1}{2}} \mathbf{s}\|^2 + \frac{1}{2} \|\mathbf{M}^{\frac{1}{2}} (\mathbf{V}^t + \Delta t \mathbf{M}^{-1} \mathbf{W}_S^u \mathbf{J} \mathbf{W}_L^s \mathbf{s})\|^2 \Big).$$

This minimization problem is quadratic, and its optimality condition leads to the following symmetric positive definite (SPD) linear system for $\mathbf{s}$:

$$\Big(\frac{1}{2} \mathbf{N}^{-1} \mathbf{W}_F^s \mathbf{W}_L^s + \Delta t \mathbf{W}_L^s \mathbf{D} \mathbf{P}^{-1} \mathbf{W}_L^{u\,-1} \mathbf{W}_F^u \mathbf{D}^T \mathbf{W}_L^s + \Delta t \mathbf{W}_L^s \mathbf{J}^T \mathbf{W}_S^u \mathbf{M}^{-1} \mathbf{W}_S^u \mathbf{J} \mathbf{W}_L^s \Big) \mathbf{s} = \mathbf{W}_L^s \mathbf{D} \mathbf{W}_F^u \mathbf{u}^* - \mathbf{W}_L^s \mathbf{J} \mathbf{W}_S^u \mathbf{V}^t. \tag{5.4}$$

Although this system is SPD, the size of the system is very large (approximately $6H \times 6H$ ignoring the relatively small number of DOFs for rigid bodies, where H denotes number of total simulation voxels), and partly dense due to the two-way coupled solid objects (Batty et al., 2007; Robinson-Mosher et al., 2008; Bridson, 2015). Given $\mathbf{u} = \mathbf{u}^* - \Delta t \mathbf{P}^{-1} \mathbf{W}_L^{u\,-1} \mathbf{D}^T \mathbf{W}_L^s \mathbf{s}$ and Eq. (5.3), I can reformulate the system above with unknown variables $\mathbf{u}$, $\mathbf{s}$, and $\mathbf{V}$ as

$$\begin{pmatrix} \frac{\mathbf{P}}{\Delta t}\mathbf{W}_F^u \mathbf{W}_L^u & \mathbf{W}_F^u \mathbf{D}^T \mathbf{W}_L^s & 0 \\ \mathbf{W}_L^s \mathbf{D} \mathbf{W}_F^u & -\frac{1}{2}\mathbf{N}^{-1} \mathbf{W}_F^s \mathbf{W}_L^s & -\mathbf{W}_L^s \mathbf{J}^T \mathbf{W}_S^u \\ 0 & -\mathbf{W}_S^u \mathbf{J} \mathbf{W}_L^s & \frac{\mathbf{M}}{\Delta t} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{s} \\ \mathbf{V} \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{P}}{\Delta t}\mathbf{W}_F^u \mathbf{W}_L^u \mathbf{u}^* \\ 0 \\ \frac{\mathbf{M}}{\Delta t}\mathbf{V}^t \end{pmatrix}.$$

From this system, I can reduce the system size by taking the Schur complement of the diagonal block matrix for the viscous stress (eliminating $\mathbf{s}$ from the system), and I obtain the following linear system for $\mathbf{u}$ and $\mathbf{V}$:

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{V} \end{pmatrix} = \begin{pmatrix} \frac{1}{\Delta t}\mathbf{P}\mathbf{W}_F^u \mathbf{W}_L^u \mathbf{u}^* \\ \frac{\mathbf{M}}{\Delta t}\mathbf{V}^t \end{pmatrix},$$

$$\mathbf{A}_{11} = \frac{1}{\Delta t}\mathbf{P}\mathbf{W}_F^u \mathbf{W}_L^u + 2\mathbf{W}_F^u \mathbf{D}^T \mathbf{W}_L^s \mathbf{N} \mathbf{W}_F^{s\,-1} \mathbf{D} \mathbf{W}_F^u.$$

$$\mathbf{A}_{12} = -2\mathbf{W}_F^u \mathbf{D}^T \mathbf{W}_L^s \mathbf{N} \mathbf{W}_F^{s\,-1} \mathbf{J}^T \mathbf{W}_S^u.$$

$$\mathbf{A}_{22} = \frac{\mathbf{M}}{\Delta t} + 2\mathbf{W}_S^u \mathbf{J} \mathbf{N} \mathbf{W}_F^{s\,-1} \mathbf{W}_L^s \mathbf{J}^T \mathbf{W}_S^u.$$

This linear system can be more efficiently solved than Eq. (5.4) because the resulting system is still SPD, the system size is much smaller (approximately $3H \times 3H$), the system matrix is sparser due to $\mathbf{V}$ treated as unknown variables even though fluid and solid dynamics are monolithically coupled (Robinson-Mosher et al., 2008; Bridson, 2015).

5.3.5 Position Correction

The divergence-free velocity fields enforce the constant volumes of fluids in the continuous setting. In practice, however, fluid volumes can change due to the spatial and temporal discretization involving numerical errors, as a volume correction method was proposed to address this issue for Eulerian fluid simulation with the level-set surface tracking (Kim et al., 2007). Given Lagrangian particles used for tracking surfaces in my method, to preserve the fluid volumes, it is necessary to enforce the uniform distribution of particles, which also addresses the gap and holes caused by non-uniform particle distributions (Bridson, 2015). While

some position-correction methods have been previously proposed (Ando and Tsuruno, 2011; Um et al., 2014), I found that damped motions of viscous fluids make the volume changes noticeable more easily and require more uniform particle distributions. To this end, I present a position-correction method using density constraints motivated by the purely Lagrangian SPH work (Macklin et al., 2014).

I define the density constraint per particle with particle density ρ_i and the rest density ρ_0 as

$$C_i = \max\left(\frac{\rho_i}{\rho_0} - 1, 0\right),$$

where ρ_i is computed by $\rho_i = m_i \sum_j w_{ij}(\mathbf{x})$ with particle mass m and position $\mathbf{x}$ and the smoothing kernel w using the traditional summation approach in SPH. Solving the constraints by correcting particle positions can be formulated as a minimization problem:

$$\Delta\mathbf{x} = \arg\min_{\Delta\mathbf{x}} \frac{1}{2}\Delta\mathbf{x}^T\mathbf{A}\Delta\mathbf{x}, \quad \text{subject to } C(\mathbf{x} + \Delta\mathbf{x}) = 0,$$

where $\Delta\mathbf{x}$ denotes the position correction, $\mathbf{A}$ mass matrix for particles, and I can compute the position correction locally following (Macklin et al., 2014) by

$$\Delta\mathbf{x}_i = -\frac{C_i\mathbf{A}^{-1}\nabla C_i^T}{\nabla C_i\mathbf{A}^{-1}\nabla C_i^T}.$$

Although correcting particle positions with this correction vector would work, many iterations are necessary in most cases. This is because this method tries to satisfy the constraints by pushing particles into the solids although these particles are again projected back to the solid surfaces using the level-set ϕ by $\mathbf{x} := \mathbf{x} - \phi\frac{\nabla\phi}{\|\nabla\phi\|}$ without caring about the particle density at the projected position. Considering this fact, to reduce necessary iterations, I allow particles farther from solid boundaries to move more distances. This approach is motivated by the mass-splitting (Tonge et al., 2012) and mass-scaling (Macklin et al., 2014). I define the scaling factor α based on the distance to solid boundaries clamping it to avoid instability as

$$\alpha_i = \text{clamp}(\phi(\mathbf{x}_i)/\theta, \alpha_{\min}, \alpha_{\max}),$$

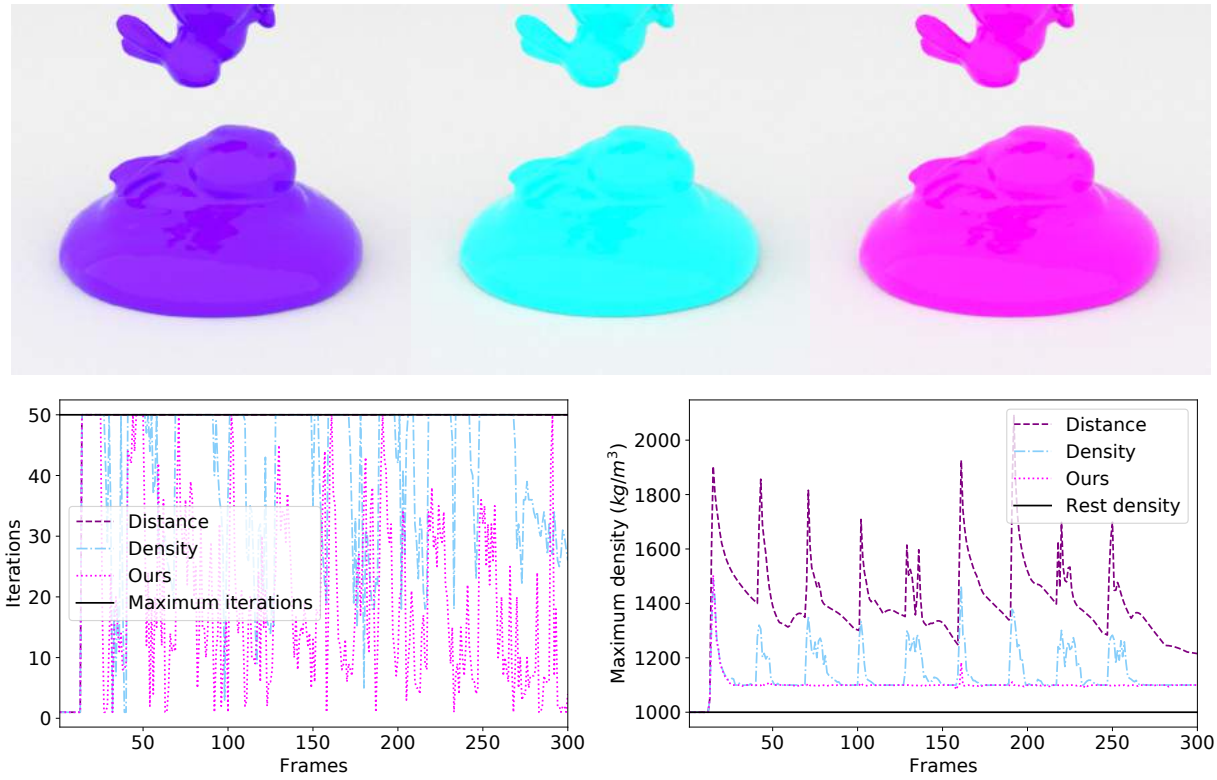


Figure 5.6: (Top) Pileup of multiple viscous bunnies simulated using the distance constraint (left), density constraint without (middle) and with (right) the scaling based on the distance to solid boundaries. (Bottom) Profiles of the iteration counts (left) and maximum density (right). The density constraint converges faster, and the scaling accelerates the convergence of the density constraints.

where θ denotes the grid width and $\alpha_{\min}$ and $\alpha_{\max}$ the minimum and maximum values for α, respectively. I typically use $\alpha_{\min} = 1.0$ and $\alpha_{\max} = 5.0$. With the scaling, I correct the particle positions by

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \alpha_i \Delta \mathbf{x}_i^l,$$

where l denotes an iteration index. The effectiveness of the scaling is demonstrated in Figure 5.6, which compares my method with the distance and density constraints based on (Ando and Tsuruno, 2011) and (Macklin et al., 2014), respectively.

5.3.6 Discussions

At the continuous formulation level, the resulting system of my method for viscous fluid dynamics is closely related to the work of (Batty and Bridson, 2008), and my method can be considered as an augmented

version of their method with geometrically consistent volume fractions for both solid boundaries and free surfaces to more accurately account for sub-grid details. In addition, I proposed the two-way coupling method to achieve interactions between highly viscous fluids and solid objects, and presented the position-correction method to address the non-uniform particle distributions.

While my formulation and its interpretation are inspired by the work of (Larionov et al., 2017), these formulations are different in that my method decouples the pressure and viscosity solves while their method couples these. As reported in (Larionov et al., 2017), these different formulations have advantages and disadvantages. For example, the decoupled approach is more efficient whereas this approach cannot faithfully reproduce coiling phenomena nor preserve surface details, and vice versa. In this work, I chose the decoupled approach mainly for efficiency. However, my position-correction method can be naturally integrated into the coupled solver (Larionov et al., 2017), and the two-way coupling method can be unified as well forming an SPD system with dense blocks for stress variables or a sparse indefinite system for pressure and velocity variables.

5.4 Results and Discussions

All the examples are executed on a Linux machine with 24-core 2.50GHz Intel Xeon and 256GB RAMs. For the viscosity solve, I use Modified Incomplete Cholesky Conjugate Gradient (MICCG) and set the convergence criterion as the infinity norm of the relative residual 1.0×10^{-10}. I used the CFL number of 3.0 with the adaptive time stepping. I performed the position correction once per frame with the termination criteria (tolerance) as 10% of the rest density. Simulation conditions and performance are summarized in Appendix D.1.

5.4.1 Volume Estimation

I compared my geometrically consistent volume evaluation method with a method that independently estimates volume fractions and the supersampling-based method (Bridson, 2015). I use a scenario, where a viscous ball is dropped onto a static, tilted dragon, as shown in Figure 5.4. With the independent volume estimation, the estimated volumes can be inconsistent over the domain, and fluid/solid and liquid/air domains can be erroneously evaluated. Consequently, the viscosity solve would generate non-physical velocity fields leading to the artifacts that particles unnaturally float in the air. With the supersampling method, although the

Table 5.1: Simulation conditions and results for Figure 5.7. Re: Reynolds number, $\hat{V}^{\infty}$: analytical terminal velocity magnitude of the ball, $V^{\infty}_{\text{oneway}}$, V^{∞}_{weak}, $V^{\infty}_{\text{strong}}$: averaged terminal velocity magnitude at equilibrium from the simulation with one-way, weak two-way, and strong two-way coupling, respectively. ϵ_{oneway}, ϵ_{weak}, ϵ_{strong}: relative errors for one-way, weak two-way, and strong two-way coupling, respectively. The gray row means that Stokes' law is invalid because of the high Reynolds number. My strong two-way coupling achieves up to approximately 10% errors and is several orders of magnitude more accurate than one-way and weak two-way coupling.

η kg/(s · m)	Re	$\hat{V}^{\infty}$ m/s	$V^{\infty}_{\text{oneway}}$ m/s	$\epsilon_{\text{oneway}}(\%)$	V^{∞}_{weak} m/s	$\epsilon_{\text{weak}}(\%)$	$V^{\infty}_{\text{strong}}$ m/s	$\epsilon_{\text{strong}}(\%)$
1.0×10^1	1.09×10^1	1.09×10^0	1.65×10^0	51.3	0.94×10^0	13.6	0.64×10^0	41.1
1.0×10^2	1.09×10^{-1}	1.09×10^{-1}	1.28×10^0	1,073.2	0.52×10^0	375.0	0.98×10^{-1}	**10.3**
1.0×10^3	1.09×10^{-3}	1.09×10^{-2}	0.57×10^0	5,173.4	0.45×10^0	4,018.5	1.01×10^{-2}	**6.9**
1.0×10^4	1.09×10^{-5}	1.09×10^{-3}	0.94×10^0	86,403.3	0.44×10^0	40,085.9	1.11×10^{-3}	**1.5**
1.0×10^5	1.09×10^{-7}	1.09×10^{-4}	0.67×10^0	616,315.8	0.43×10^0	396,816.0	1.12×10^{-4}	**2.3**

volume estimation can be consistent over the domain, the estimated volumes are not accurate enough since this approach relies on volume estimation at specific points only. Consequently, similar to the independent volume estimation, particles can be unnaturally left in the air. In contrast, my method enforces the volume consistency over the domain achieving more accurate estimations and does not suffer from the artifacts with a little additional cost (approximately 13% more costly for the volume computation than the inconsistent version and the supersampling method leading to only about 5% overhead for the total computation time).

5.4.2 Two-Way Fluid-Solid Coupling

To validate the accuracy of my strong two-way coupling for viscosity, I first experimented with a simple scenario, where a solid ball is falling inside of viscous fluids, so that analytical solutions of the solid velocity can be computed. I use fluid viscosity values $\eta = 1.0 \times 10^1, 1.0 \times 10^2, 1.0 \times 10^3, 1.0 \times 10^4$, and 1.0×10^5 kg/(s · m) with fluid density $\rho_f = 1.0 \times 10^3$ kg/m^3, solid density $\rho_s = 3.0 \times 10^3$ kg/m^3, and the radius of the solid ball $r = 5.0 \times 10^{-2}$ m. I compare my method with one-way coupling from solid-to-fluid, weak two-way coupling (viscosity solve followed by an implicit solid velocity update), and the analytical solution derived from the Stokes' law. As for pressure forces, I achieve strong two-way coupling based on the works of (Batty et al., 2007; Ng et al., 2009). Figure 5.7 shows the comparison executed with $\eta = 1.0 \times 10^3$ kg/(s · m). For one-way and weak two-way coupling, the solid ball is treated as prescribed (or kinematic) object in the viscosity solve, giving rise to incorrect fluid and solid dynamics even though viscosity force is applied in the case of weak two-way coupling. On the other hand, the strong two-way coupling method appropriately handles the interplay with viscosity forces between the fluids and the solids in

the viscosity solve, leading to the terminal solid velocities close to the analytical solution. Although iterative refinements are possible to improve the accuracy with the weak coupling as implied in (Akbay et al., 2018), in this scene, the computation time for viscosity solve with one-way, weak two-way, and strong two-way coupling is almost the same and occupies approximately 25–40% of the total simulation time. Thus, I believe that the iterative refinement with weak two-way coupling will become more costly than my strong two-way coupling.

Table 5.1 summarizes the simulation conditions and results. The velocities of the solid ball simulated with the one-way and weak two-way coupling methods significantly deviate from the analytical solutions whereas my method generates the solid velocities very close to the analytical solutions (relative errors are up to around 10%) except for the case of $\eta = 1.0 \times 10^1$ kg/(s $\cdot$ m), where the Stokes' law is not valid due to the high Reynolds numbers.

I also compared my method with one-way and weak two-way coupling in a more complex scenario, where multiple solid bunnies are dropped onto fluid volumes with spatially varying viscosity values from $\eta = 1.0 \times 10^1$ to 1.0×10^8 kg/(s $\cdot$ m), as shown in Figure 5.5. One-way and weak two-way coupling methods do not generate plausible motions nor sufficiently reflect the different viscosity values to the dynamics, whereas my method produces natural behaviors of solid objects as expected with different viscosity values.

5.4.3 Position Correction

To demonstrate the effectiveness of my position-correction method, I experimented with a scene, where a bulk of viscous fluids is successively compressed by prescribed, solid circular plates with multiple holes, as shown in Figure 5.8. In this scene, I compared my method with previous approaches using no position correction and position correction based on distance constraints (Ando and Tsuruno, 2011). The previous approach without any position corrections easily loses the fluid volumes. While the distance-based position correction better preserves the fluid volumes, still some volumes are lost because this approach is not designed to preserve particle density (and volumes). By contrast, my method directly enforces the uniform particle distributions leading to particle densities closer to the rest density (see particles color-coded based on their densities in Figure 5.8) and preserving the fluid volumes. This enables fluid volumes to be sufficiently pushed by the solid plates and come out from the holes, reaching the top of the plates.

5.4.4 Complex Examples

Figure 5.9 demonstrates three gears two-way coupled with viscous fluids. In this scene, only one rotational DOF for each gear is effective in the viscosity solve, and two rotational and three translational DOFs are eliminated from the system. I note that the two-way coupling with viscosity forces accounts for the dragging effects, which can accelerate and decelerate the angular velocities of the gears.

Figure 5.10 demonstrates complex two-way interactions between multiple solid bunnies and viscous fluids in a rotating drum. Because of the two-way fluid-solid coupling followed by the collision handling between solid objects, interactions between the solid bunnies and the drum in viscous fluids are naturally simulated.

Figure 5.11 demonstrates the yogurt smoothie interacting with dropped fruits. The yogurt smoothie is pushed by the dropped strawberry and thus overflows from the cocktail glass, as observed in the real phenomena.

5.4.5 Discussions, Limitations, and Future Work

Volume fractions. In the viscosity solve, I account for the sub-grid geometry with volume fractions based on the variational approach (Larionov et al., 2017). In the pressure solve, it is known that using the cut-cell method for the fluid-solid interface (Ng et al., 2009) and ghost fluid method for the liquid-air interface(Gibou et al., 2002) generally gives more accurate results. However, it is not straightforward to employ these approaches in the viscosity solve, e.g., because the cut-cell methods cannot consistently define the area fractions for velocity components due to changing flux directions with the diagonal and off-diagonal components of the viscous stress. I tested the cut-cell method with a fixed flux direction respecting the diagonal viscous stress components. Although I achieved better accuracy suppressing non-physical oscillations of the solid ball velocities in the scenario shown in Figure 5.4, I found that neglecting the flux directions of the off-diagonal components leads to popping visual artifacts of particles. Thus, it would be necessary to investigate how to consistently account for the sub-grid geometries to achieve higher accuracy without the artifacts.

Boundary condition. When I consider viscosity formulations with solid objects, it is common to use the no-slip boundary condition because boundary layers, where viscous forces become dominant, are formed at the proximity of solids in reality. Although this approach works well for fluids with relatively high viscosity

values, when fluids are less viscous, the effect of the no-slip boundary condition is magnified up to the scale of simulation cells (which are generally much larger than the thickness of the actual boundary layers) exhibiting excessively viscous behaviors on the boundaries. Using higher resolutions to resolve these issues drastically increases the simulation cost and thus is impractical. One approach to simulating less viscous fluids with solid boundaries is to use fluid velocities instead of solid velocities at the boundaries. In this case, however, the stability of the viscosity solve is not guaranteed, and I encountered the stability issues in my early experiments. Although the use of the free-slip condition or Navier-slip boundary condition for friction is suggested in (Bridson, 2015), formulations and implementation become more complex. It would be worthwhile to explore efficient approaches to enforcing these boundary conditions for fluid simulation with a wide range of viscosity values.

Density constraint. The density constraint can resolve the compression of fluids as shown in Figure 5.8. However, since the density constraint does not attract particles each other, when particles are separating from the others, fluid volumes would suffer from bumps and holes at the limit of the particle resolution. Although attraction forces can be used, I found that the attraction forces cause particle clustering, which is known as the tensile instability in SPH. To address this issue, techniques, such as particle split and merge operations (Narain et al., 2010), particle sampling (Yue et al., 2015), a narrow band approach using the level-set (Sato et al., 2018) might be helpful.

Unified solve. While I preferred to decouple pressure and viscosity solves for efficiency, there are certain scenarios, where pressure-viscosity coupled solvers are preferable due to, e.g., capability of coiling, better energy preservation, and more surface details (Larionov et al., 2017). In addition, the second pressure solve in the decoupled approach would affect the results of the viscosity solve, leading to artificial melting artifacts, e.g., the right most bunny gradually sinking in fluids regardless of the very high viscosity value (=1.0×10^8 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$). Similarly, since my method separately addresses fluid-solid and solid-solid interactions, intensive solid-solid collisions would cause various negative effects, e.g., oscillations of velocity fields and the loss of fluid volumes, both of which can be observed in Figure 5.5. For future work, I plan to develop a solver that can simultaneously handle pressure and viscosity solves and fluid-solid and solid-solid interactions for robust, accurate, and consistent simulations.

5.5 Conclusions

I proposed an augmented viscous fluid solver to simulate a wider range of viscous fluid scenarios. My method offered three key contributions. 1) To avoid the dangling artifacts, I presented a geometrically consistent volume estimation method. 2) I also proposed a new two-way coupling technique that can handle interactions between fluids and solid objects with *viscosity* forces. My two-way coupling formulation can be seamlessly integrated into the implicit viscosity solve and produces a sparse SPD system monolithically unifying the dynamics of viscous fluids and solid objects. 3) I presented a position-correction method that enforces the uniform distributions of particles to address the volume loss issues, and accelerated the convergence of position corrections with the scaling scheme based on the distance to solid objects. I verified the accuracy of my method by comparing the results with the analytical solutions and demonstrated the effectiveness of my solver in various challenging scenarios.

Acknowledgements

This work is supported in part by National Science Foundation and Elizabeth Stevinson Iribe Chair Professorship. I would like to thank anonymous reviewers for their valuable suggestions and comments.

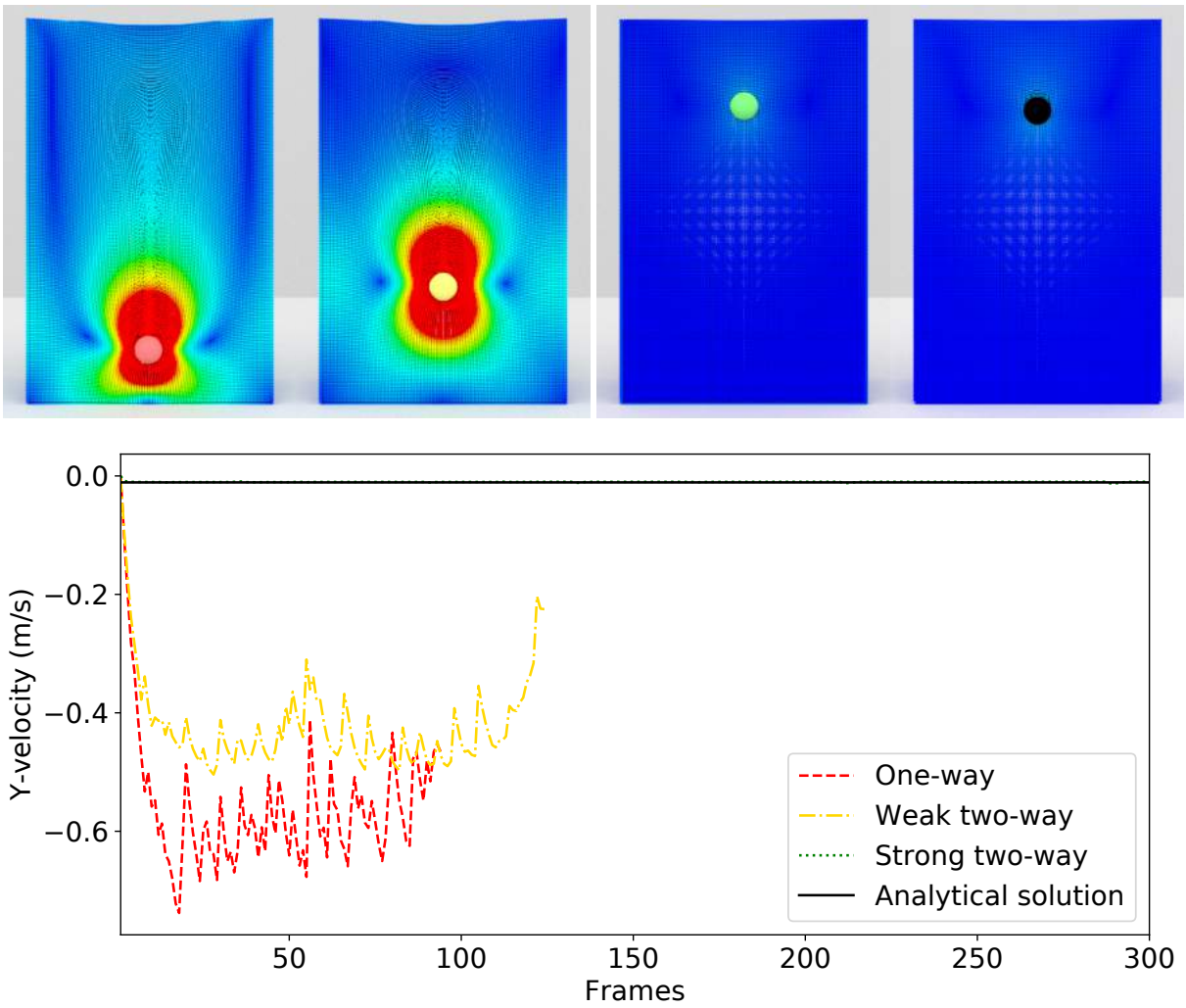


Figure 5.7: A solid ball falling inside of fluids with viscosity value $\eta = 1.0 \times 10^3\ \mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. Fluids simulated with one-way and weak two-way coupling cannot sufficiently support the solid ball due to the incorrect viscosity forces while my strong two-way coupling method successfully supports the solid ball. (Bottom) A profile for y-velocity of the solid balls with the analytical solution. The resulting solid velocities with one-way and weak two-way coupling significantly deviate from the analytical solution, whereas the solid velocities given with my strong two-way coupling are in good agreement with the analytical solution.

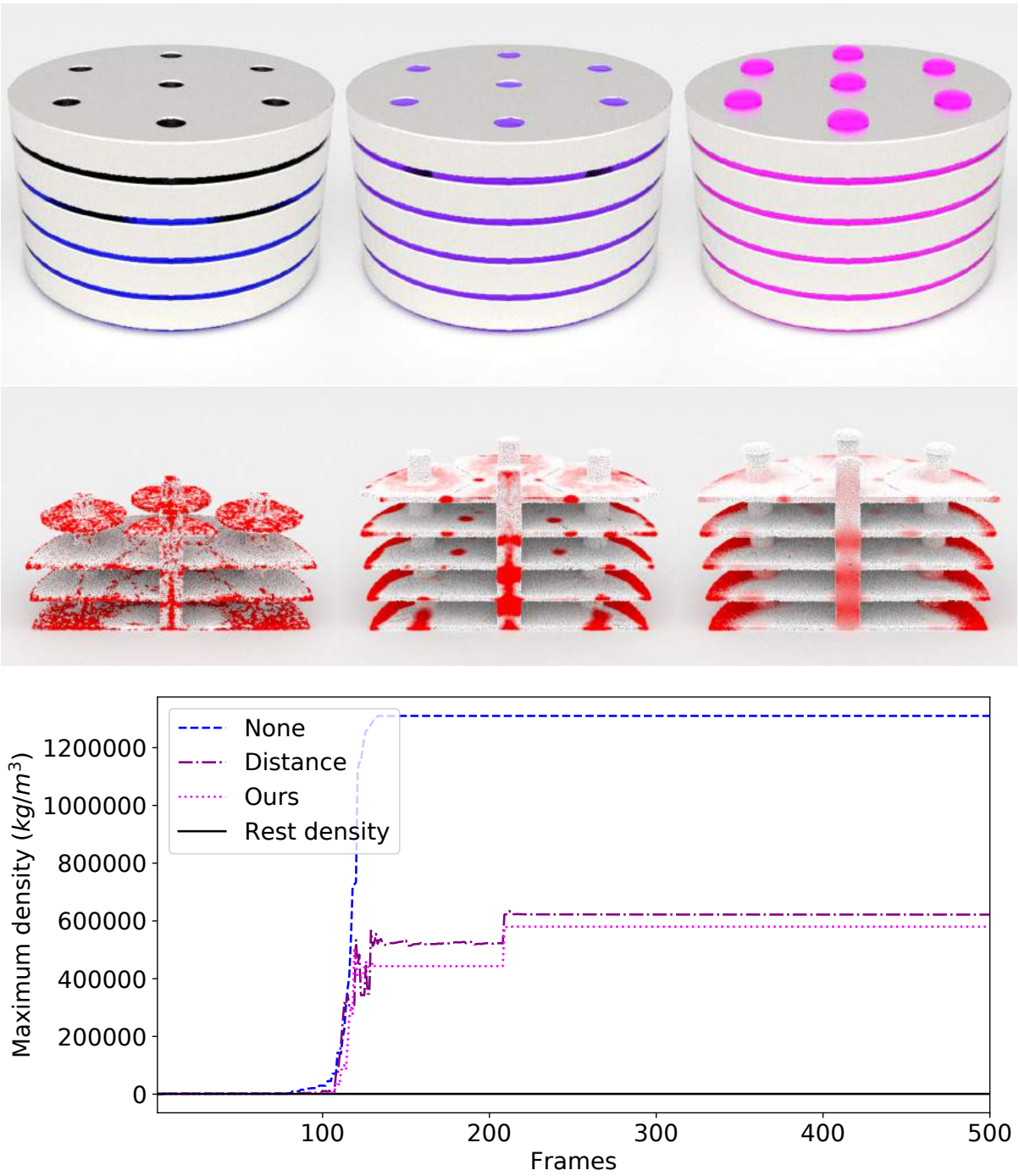


Figure 5.8: (Top) A viscous fluid volume successively compressed by prescribed circular plates with several holes. From left to right, no position correction, distance-based position correction, and my method for surface rendering and particle view with color coding (white: low density, red: high density). Because of the density constraint, my method can better preserve the volume of the viscous fluids reaching the top of the plates while other approaches fail to reach the top due to the volume loss. (Bottom) Profile of the maximum particle density, which indicates the inverse of local volumes. Compared to other approaches, my method preserves the density closer to the original one.

Figure 5.9: Three gears interacting with viscous fluids with $\eta = 1.0 \times 10^1, 1.0 \times 10^2$ and 1.0×10^3 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$ from left to right. Different viscosity values induce distinct fluid behaviors and solid rotations.

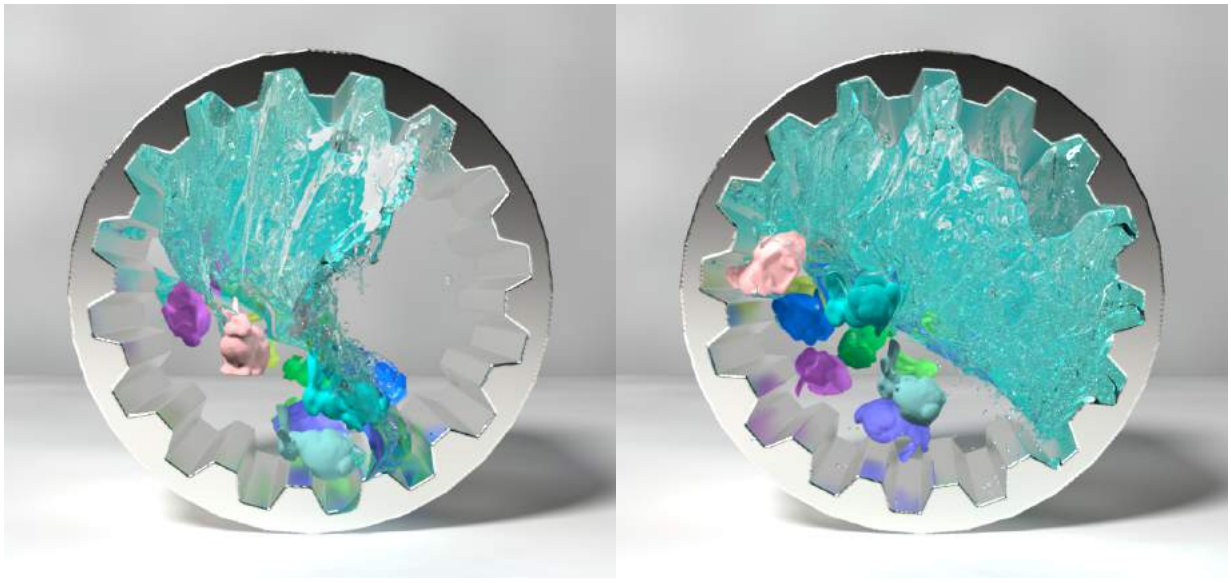


Figure 5.10: Multiple solid bunnies interacting with viscous fluids in a rotating drum. Simultaneous one-way (between the bunnies and rotating drum) and two-way (between the bunnies) solid collisions can be addressed by combining my fluid solver with a rigid body solver. Front and back sides are clipped for visualization.

Figure 5.11: (Left) Yogurt smoothie overflowing from a cocktail glass because of the dropped strawberry. (Right) Real yogurt smoothie.

CHAPTER 6: Video-Guided Real-to-Virtual Parameter Transfer for Viscous Fluids

6.1 Introduction

Fluids are ubiquitous and common – encountered in various aspects of our everyday lives. Examples of these materials include water, milk, honey, machinery oil, molten chocolate, paint, and shampoo. These liquids have different properties and exhibit distinct behaviors. One key factor that determines fluid properties and behaviors is viscosity, as this can be realized from the Reynolds number, which consists of viscosity parameters and characterizes flow patterns of fluids. For example, fluids with low viscosity values flow vividly generating turbulence and splashes, whereas highly viscous fluids exhibit damped motions and characteristic rotational behaviors, such as buckling phenomena. While many of previous works have focused on inviscid fluids, several researchers have attempted to more accurately simulate the dynamics of highly viscous fluids and improved the visual fidelity with physically-based viscosity models (Carlson et al., 2002; Batty and Bridson, 2008; Larionov et al., 2017; Bergou et al., 2010; Batty et al., 2012; Zhu et al., 2015).

While physically-based approaches can effectively simulate viscous fluids based on the physical properties of fluids, one known challenge is that it can be very difficult, time-consuming, and tedious to choose appropriate parameters to generate desirable fluid behaviors, e.g., approximating behaviors of viscous fluids observed in the real world. If physical parameters are not ideal, these approaches would generate visually disconcerting results, which negatively impact our sense and recognition of the fluid materials and dampen

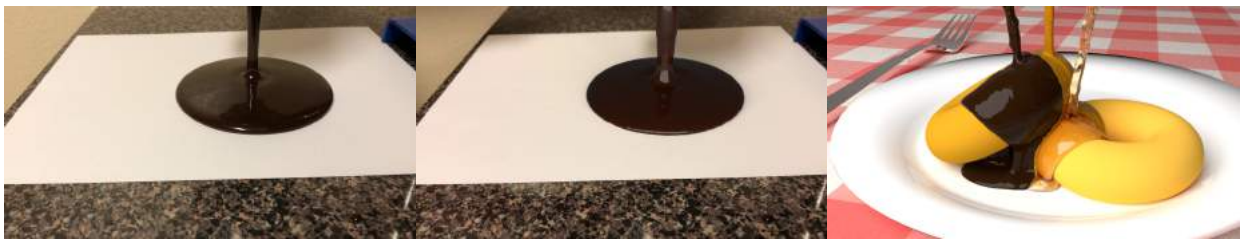


Figure 6.1: My framework identifies the set of physical variables and viscosity parameters automatically from example videos capturing fluid flows in the real world (left) by approximating the flows with viscous fluid simulation (middle). The identified physical values and parameters can then be used to simulate viscous fluids in a new scenario, preserving the style of the fluid flows in the example videos (right).

our experience in various applications, such as video games, movies, and virtual reality. Even worse, such parameters would cause simulation failure leading to unpredictable results. Consequently, it is necessary to manually tune parameters through laborious trial-and-error processes until satisfactory visual results are obtained. In practice, fluid simulation can take several hours or more, requiring many hours of waiting time to check intermediate results. Thus, such manual parameter-tuning is beyond practical.

One possibility to select appropriate parameters for fluid simulation is to adopt material parameters listed in a book or measured in the real world (e.g., using a viscometer and a rheometer). In general, however, viscosity values of most of fluid materials are not available at hand, and such instruments are not widely available for personal use, as mentioned in (Nagasawa et al., 2019). In addition, fluid simulation is one way to approximate the behaviors of real complex fluid flows using a simplified mathematical model of physics to make the simulation tractable, and it is known that different fluid simulation methods often lead to distinct simulation results even though the same governing equations are solved under the same simulation setting and physical parameters (Um et al., 2017). As such, there is no guarantee that fluid simulation with the viscosity values found in a handbook or measured with a viscometer can generate fluid behaviors similar to those observed in the real world, except for few limited, ideal situations. Furthermore, since some fluid simulation methods are devised to improve the efficiency, robustness, and capability based on heuristics, these methods may not have physical parameters corresponding to their counterparts in the real world. From a viewpoint of artists, physical parameters are not necessarily intuitive enough to generate their conceived fluid effects because changes in physical parameters modify fluid flows in a complex and unpredictable way, and the same material parameters can lead to different behaviors depending on simulation scales. Instead, one possible approach is to use examples of desired fluid behaviors to infer appropriate parameters based on the given observed examples.

In this chapter, I propose a new material parameter optimization framework for automatic parameter identification for fluids with example videos captured in the real world. My framework takes as input a video capturing real fluid flows and extracts positional information of fluids from the example video for a reference. Then, I identify the set of physical values and viscosity parameters by minimizing the differences between the example video and fluids simulated with my viscous fluid solver in an iterative process. Since it is challenging to accurately reconstruct 3D fluid information from 2D videos, I measure the differences of the example data and my simulation results in the 2D screen space by projecting my 3D simulation results onto the screen space. Because of the 3D simulation analysis in the iterative process, the results of the

forward simulation with the identified parameters allow us to infer hidden physical quantities of fluids in the videos. Furthermore, the identified parameters can be used in completely new scenarios while preserving the styles of the fluid behaviors in the example videos. To show the effectiveness of my framework, I validate the identified parameters in various scenarios, infer hidden physical quantities, and demonstrate the parameter transfer from the real world to virtual environments.

In summary, my main contributions and key results include:

- **A parameter optimization framework** that identifies the viscosity parameters for fluids based on example videos captured from real-world fluid phenomena, inferring hidden physical quantities of fluids.
- **Screen-space evaluation** that allows for measuring differences between the example data and simulation results without reconstructing 3D data.
- **Parameter transfer from real to virtual environments**. It introduces a new data-driven approach for fluid animation and enables us to reproduce fluid behaviors in the virtual environment, preserving the observed fluid properties in the real world.

To the best of my knowledge, my framework is the first method for identifying material parameters of fluids, and Figure 6.1 demonstrates the effectiveness of my framework.

6.2 Related Work

Fluid simulation has been a major research topic of significant interest in computer graphics, and various techniques have been proposed. In this section, I focus my discussion on previous works closely related to my method. Later, I also discuss several works on material parameter estimation.

6.2.1 Viscous Fluid Simulation

Viscous fluids exhibit behaviors different from inviscid fluids, and reproducing their characteristic behaviors has been required over years for various applications. In the Eulerian approach, Stam (Stam, 1999) developed a stable fluid method using implicit integration with the Laplacian form of viscosity for fluids without free surfaces. Later, Carlson et al. (Carlson et al., 2002) extended the method with implicit viscosity integration for fluids with free surfaces, compromising the accurate handling of rotational motions at the free surfaces. Rasmussen et al. (Rasmussen et al., 2004) augmented the implicit Laplacian-based formulation

with explicitly integrated off-diagonal components to account for the rotational behaviors while sacrificing the robustness of their solver. Batty and Bridson (Batty and Bridson, 2008) proposed a fully implicit viscosity integration scheme for the full form of viscosity to improve the simulation accuracy in the free surface handling, and this approach was extended for adaptive tetrahedral meshes (Batty and Houston, 2011) and octree data structures (Goldade et al., 2019), and for two-way solid-fluid coupling (Takahashi and Lin, 2019a). Larionov et al. (Larionov et al., 2017) proposed a pressure-viscosity coupled solver to further improve the accuracy in the free surface handling. Recently, Kim et al. (Kim et al., 2019) proposed an efficient deep-learning-based framework to interpolate simulation results using different viscosity values. Unlike these approaches for 3D volumes, Vantzos et al. (Vantzos et al., 2018) proposed an efficient two-dimensional approach to simulating viscous thin films.

To simulate more general fluids, e.g., viscoelastic fluids and non-Newtonian fluids, various approaches have been also proposed. Goktekin et al. (Goktekin et al., 2004) presented a method for simulating viscoelastic fluids with extra elastic forces. To handle fluids with a variety of properties in a unified way, material point methods have been widely adopted with some specialized extensions for snow (Stomakhin et al., 2013), foams (Yue et al., 2015), melting solids (Stomakhin et al., 2014), elastoplastic solids (Gao et al., 2017; Fang et al., 2019), and granular materials (Klár et al., 2016; Daviet and Bertails-Descoubes, 2016; Yue et al., 2018). Recently, to better approximate the behaviors of blended fluid materials, Nagasawa et al. (Nagasawa et al., 2019) proposed a parameter blending scheme using the method of (Yue et al., 2015). While these approaches allow us to simulate a wide range of materials, relations between simulation parameters in their constitutive laws and material parameters are complicated. Thus, in this chapter, I focus on purely Newtonian viscous fluids.

In the Lagrangian setting, one commonly used approach to simulating viscous fluids is based on Smoothed Particle Hydrodynamics (SPH), and various approaches have been proposed to improve the efficiency and robustness. Takahashi et al. (Takahashi et al., 2015) proposed an implicit viscosity integration to improve the robustness compared to explicit integration, adopting the method of (Batty and Bridson, 2008). To further improve the efficiency, Peer et al. (Peer et al., 2015) presented a different implicit viscosity integration model with prescribed gradient, compromising the physical consistency, and this approach was extended to improve the diffusivity of the vorticity (Peer and Teschner, 2017) and to support a wider range of viscous fluid behaviors (Bender and Koschier, 2016). Recently, Weiler et al. (Weiler et al., 2018b) presented a robust and efficient implicit viscosity formulation while achieving physical consistency. To handle fluids with various

properties in a unified way, Barreiro et al. (Barreiro et al., 2017) proposed using conformation constraints within the position-based dynamics framework.

Unlike these approaches based on SPH, some works simulated viscous fluids by formulating particle interactions using spring forces between particles (Clavet et al., 2005; Takahashi et al., 2014). Taking advantages of Lagrangian discretization, several specialized techniques based on simplicial complexes have been proposed to simulate viscous threads and sheets (Bergou et al., 2010; Batty et al., 2012; Zhu et al., 2015).

In the fluid simulation literature, a variety of simulation methods have been proposed to simulate viscous fluids. However, how to automatically select appropriate parameters for simulation methods has not yet been investigated, and I contribute to this topic.

6.2.2 Fluid Capturing

Capturing fluids has been a challenging problem over the decades. One reason is that fluids do not have their rest shapes, and this fact makes it unreasonable to assume predefined shapes or deformations from specific shapes, which can be effectively used in capturing the dynamics of rigid bodies (Monszpart et al., 2016) and deformable objects (Wang et al., 2015). In addition, the appearance of fluids can be easily and significantly affected by surrounding environments, e.g., due to light scattering, absorption, reflection, and refraction, making fluid capturing even more challenging.

An early work to model fluid volumes was proposed by Ihrke and Magnor (Ihrke and Magnor, 2004) and Hasinoff and Kutulakos (Hasinoff and Kutulakos, 2007). They reconstructed fluid volumes by solving a least squares problem, penalizing differences between numerically computed pixel intensity and observed intensity. These approaches were extended to avoid blurry, reconstructed volumes by transferring the appearance of fluid volumes (Okabe et al., 2015). For the dynamic 3D volume reconstruction, several researchers made use of volume representations, similar to tomography. Atcheson et al. (Atcheson et al., 2008) modeled dynamic gaseous volumes based on information captured with multiple cameras. Gregson et al. (Gregson et al., 2012) focused on fluid mixing based on dye concentrations. A similar minimization approach was employed to reconstruct 3D liquid surfaces, but not volumes, with submerged checker board patterns (Morris and Kutulakos, 2011). The main focus of these works are on modeling fluid geometry, and velocities of the fluids are not inferred or roughly estimated with an assumption on the rotational symmetry.

To compute more accurate velocity fields based on data captured from real-world phenomena, e.g., videos, several researchers have proposed methods that combine fluid simulation with iterative inversion. Wang et al. (Wang et al., 2009) reconstructed not only fluid volumes but fluid velocities from fluid videos, which were captured using synchronized stereo cameras with dyed fluids. Li et al. (Li et al., 2013) recovered water surfaces and their velocities by combining the shallow water simulation with water surfaces reconstructed using a shape reconstruction method based on shading. Gregson et al. (Gregson et al., 2014) proposed a velocity reconstruction framework based on an optical flow method with physics regularizer terms similar to (Corpetti et al., 2002; Chen et al., 2016), combining tomographic 3D volume information captured with the method of (Gregson et al., 2012), and this framework was augmented to achieve the velocity reconstruction from a single-view video (Eckert et al., 2018). Recently, Zang et al. (Zang et al., 2019) proposed a tomographic reconstruction algorithm for time-varying deforming objects, capturing both of the volumes and deformation fields.

In the physics literature, researchers often utilized sophisticated hardware to directly capture the fluid volumes or velocity fields. One popular approach is Particle Image Velocimetry (PIV), and a good overview for PIV is given in (Grant, 1997). PIV injects tiny particles into fluids, illuminates the particles with a sheet of laser light, and then estimates the particle movements and fluid velocities. In the graphics literature, Xiong et al. (Xiong et al., 2017) proposed a new PIV algorithm that colors particles based on their depth to track 3D velocity fields with a single camera.

While various algorithms have been proposed for fluid capturing, these algorithms typically require a sophisticated setup, such as synchronized cameras, dyed fluids, and laser device. Thus, in my framework, I avoid such setup and use a commonly available device, smartphone, for fluid capturing. However, I note that these capturing techniques are orthogonal to my goal and can be easily incorporated into my framework.

6.2.3 Material Parameter Estimation

In physically-based simulations, choosing simulation parameters is one of the most critical steps to generate visually plausible results or even to perform stable simulations. Because of the importance and difficulty in tuning physical parameters, various researchers have attempted to automate this process.

One commonly used approach for material parameter estimation is to find optimal parameters that generate behaviors close to example data, e.g., captured in the real world, and this approach has been extensively adopted in the literature, especially for deformable solids (Gerlach and Matzenmiller, 2007).

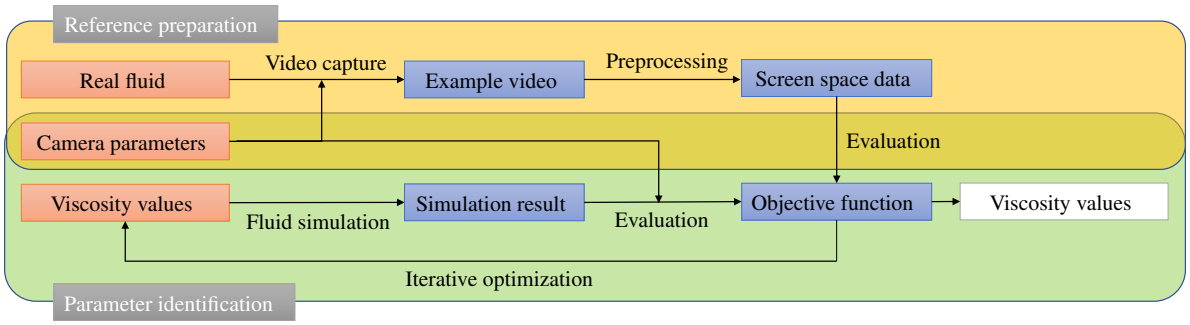


Figure 6.2: Overview of my parameter identification framework. My framework consists of two stages: reference preparation and parameter identification. In the reference preparation stage, I capture a video of real fluid flows and preprocess the video to extract positional information of the fluid. In the parameter identification stage, I iteratively perform fluid simulation, project simulated fluids onto the screen space, and evaluate objective functions with the extracted fluid data. Finally, my framework outputs identified viscosity values.

Pai et al. (Pai et al., 2001) proposed a method for acquiring material parameters from interactions with deformable objects via robotic measurement facility. Becker and Teschner (Becker and Teschner, 2007a) proposed a framework to optimize elasticity parameters with linear Finite Element Method based on the relation between the initial undeformed geometry and applied forces. Lee and Lin (Lee and Lin, 2012) also presented a framework to identify material parameters using FEM simulation by minimizing the distances between surface nodes from the simulation and reference. Bickel et al. (Bickel et al., 2009) proposed a method for optimizing the material properties of deformable objects with deformation measurements taken from real-world experiments. Later, Bickel et al. (Bickel et al., 2010) used their techniques to fabricate deformable objects that have their desirable properties. These material parameter optimization techniques were further extended using model reduction to improve the efficiency (Xu et al., 2015). Xu and Barbič (Xu and Barbič, 2017) presented an optimization framework for damping coefficients to improve the behaviors of deformable objects. Yan et al. (Yan et al., 2018) presented an inexact descent approach to accelerating the parameter optimization of elastic materials. Deformation measured in the real world was also used for the parameter identification for clothing (Wang et al., 2011; Miguel et al., 2012; Clyde et al., 2017) and human body (Pai et al., 2018). In sound rendering, sound captured from various materials was also used as reference data to optimize the audio material parameters (Ren et al., 2013).

While some approaches take example data from the real world for references, these references can be prepared by users. Twigg et al. (Twigg and Kačić-Alesić, 2011) proposed an optimization method that finds a

user-specified shape under gravity. A similar optimization approach was employed and extended to handle frictional contact for hair (Derouet-Jourdan et al., 2013) and shells (Ly et al., 2018). Li et al. (Li et al., 2014) presented a space-time optimization framework that simultaneously optimizes the dynamics and material parameters of subspace deformable objects.

Several researchers also proposed material parameter identification methods based on images and videos to avoid using specialized equipment to estimate deformations and forces. Wang et al. (Wang et al., 2015) proposed a material parameter optimization approach by combining expectation maximization method and Nelder-Mead method. Yang and Lin (Yang and Lin, 2016) identified material properties for deformable objects from a few images with the particle swarm method. These material parameter optimization techniques with videos taken in the real world are also applied to cloth (Bhat et al., 2003; Yang et al., 2017, 2016), hair (Hu et al., 2017) and rigid bodies (Bhat et al., 2002; Monszpart et al., 2016).

Although various attempts have been made to facilitate the parameter tuning and selection, little research has been conducted for fluids. In this chapter, I address this problem, and propose perhaps the first method for identifying the material parameters for fluids using captured video data.

6.3 Overview

My goal is to identify material parameters of fluids, with which a viscous fluid simulator can generate fluid behaviors as close as possible to the example data captured from real-world phenomena. Figure 6.2 illustrates my material parameter identification framework. My framework consists of two stages: reference preparation stage and parameter identification stage. In the reference preparation stage, my framework takes as input example videos captured from real-world fluid phenomena. Then, I preprocess the videos and extract positional data of the fluid so that these data are amenable in the following parameter identification stage. The parameter identification stage is an iterative process and takes initial or refined material parameters as input. In this stage, I first perform forward fluid simulations with the material parameters to obtain simulation results. Next, I project the simulation results onto the screen space with the camera parameters used to capture the example videos, and then evaluate my objective functions, which measure the differences between the example data and projected simulation results. My framework iteratively refines the material parameters and finally outputs identified material parameters.

In my framework, a viscous fluid solver is iteratively used within the parameter identification stage, and the identified material parameters (which can be used in different scenarios) are for the viscous fluid solver. While my framework is not restricted to a specific fluid solver, for self-containedness, I first briefly review my viscous fluid solver in § 6.4. The details of my material parameter identification framework are described in § 6.5.

6.4 Viscous Fluid Solver

The dynamics of viscous fluids can be described by the incompressible Navier-Stokes equations given by

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \mathbf{s} + \frac{1}{\rho}\mathbf{f}, \tag{6.1}$$

$$\mathbf{s} = \eta\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right), \tag{6.2}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{6.3}$$

where t denotes time, $\frac{D}{Dt}$ material derivative, $\mathbf{u}$ velocity, ρ density, p pressure, $\mathbf{s}$ symmetric viscous stress tensor, $\mathbf{f}$ external force, and η dynamic viscosity. I include the gravity force and surface tension force as external forces. To advance the simulation step, I first address the advection term with the affine particle-in-cell (APIC) approach (Jiang et al., 2015), add external forces, and then handle pressure and viscosity terms simultaneously.

I address the pressure and viscosity terms in a unified and implicit manner as

$$\frac{\mathbf{u}^{t+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \mathbf{s}^{t+1}, \tag{6.4}$$

$$\mathbf{s}^{t+1} = \eta\left(\nabla\mathbf{u}^{t+1} + \left(\nabla\mathbf{u}^{t+1}\right)^T\right), \tag{6.5}$$

$$\nabla \cdot \mathbf{u}^{t+1} = 0, \tag{6.6}$$

where $\mathbf{u}^*$ denotes intermediate velocity after advection and external force steps, and Δt time step size. To solve the unified pressure-viscosity problem, I discretize it based on the variational principle (Larionov et al., 2017) using the volume computation method described in (Takahashi and Lin, 2019a).

While the dynamic viscosity η can be spatially and temporally varying, in this chapter, I mostly focus on the static case for η as most of real Newtonian fluids hold a uniform property over the fluid volume, and this make the parameter identification and validation of the identification results more tractable. In the next section, I aim to identify the viscosity parameter η based on a given example video.

6.5 Viscosity Parameter Identification

My method identifies the viscosity parameters of fluids by minimizing the differences between example data captured from real-world phenomena and fluids simulated with my viscous fluid solver. While multiple formulations can be considered for this parameter identification problem, e.g., with soft constraints, to invalidate physics violations and undesirable local minima (Yan et al., 2018), I formulate the problem with hard constraints as the following constrained space-time optimization problem:

$$\eta = \underset{0 \leq \eta}{\arg\min}\, E, \tag{6.7}$$

$$E = \sum_f \omega_f E_f \quad \text{subject to } \mathbf{Q}_{f+1} = F(\mathbf{Q}_f), \tag{6.8}$$

where E denotes an objective function, which measures the differences between example data and the simulated fluids, ω weighting coefficients for each frame, $\mathbf{Q}$ a state variable for fluids, F a function for the forward simulation, and frame index $f = 0 \ldots N-1$, where N denotes the number of frames considered in the optimization.

6.5.1 Objective Function

To use videos as a reference for the parameter identification, it is necessary to extract some information on fluids, such as fluid geometry, that can be compared with results of 3D fluid simulations. In the literature, some works attempted to reconstruct 3D fluid geometry and velocity from videos, e.g., (Wang et al., 2009; Okabe et al., 2015; Li et al., 2013). However, these approaches typically require a complex equipment setup, such as synchronized multiple cameras, depth sensors, and/or dyed liquid; or they need to restrict fluid motions because it is very challenging to reconstruct 3D fluid data from videos which include 2D information only (i.e., 3D information is already lost). Since there are multiple 3D fluid configurations, which lead to similar 2D fluid configurations on the screen, the 3D data reconstruction from 2D videos is ambiguous, i.e.,

this problem is under-determined. Additionally, fluids generally have no preferred shape, and thus it is not reasonable to consider the rest shape or deformations from the rest shape, making it difficult to capture the fluid geometry, unlike rigid and deformable bodies. Furthermore, the appearance of fluids can be easily and significantly changed due to the optical properties of fluid surfaces, e.g., with light scattering, absorption, reflection, and refraction, and thus it is very difficult to obtain the reliable 3D fluid data from videos.

Therefore, I eschew reconstructing 3D fluid data and instead measure the differences between the example data and results of 3D simulation on the 2D screen space (with the same size as example videos). In my framework, I evaluate the differences in terms of the fluid geometry in the 2D space (i.e., as a silhouette) and define my objective function for frame index f as

$$E_f = \frac{1}{2M}(\mathbf{g}_f - \hat{\mathbf{g}_f})^T \mathbf{C}_f (\mathbf{g}_f - \hat{\mathbf{g}_f}) \tag{6.9}$$

where M denotes the total count of pixels in the screen space, $\mathbf{C}$ a diagonal coefficient matrix to focus on specific domains in the screen space, $\mathbf{g}$ and $\hat{\mathbf{g}}$ denote silhouette obtained from simulation results and extracted from the example videos, respectively. I use binary values for $\mathbf{g}$ and $\hat{\mathbf{g}}$, and define them at each pixel in the 2D screen space.

6.5.2 Fluid Video Capturing

For the reference silhouette $\hat{\mathbf{g}}$, I first capture example fluid videos. While there are various ways to capture the fluid videos, it is important to adopt a setup, which can be easily prepared and used to capture different viscous fluid materials while minimizing sources of errors (e.g., human interventions and gaps between the simulation and real fluid flows) as much as possible. Although one intuitive setup would be to pour liquids from a container, I found that liquid pouring is not ideal because it requires some human interventions (i.e., manipulations of the container) and forms very thin fluid sheets near the edge of the container, causing too strong surface tension forces which dominate viscosity forces. Given these, I prepare a simple setup, where viscous fluids flow from the hole at the bottom of the container due to the gravity, as shown in Figure 6.3.

In my work, I use a normal smartphone, iPhone 8, and capture the fluid flows with the resolution of 1280×720 at 30 fps from a single view. I fix the camera positions, calibrate the camera to obtain intrinsic

Figure 6.3: Setup for capturing a single-view video for behaviors of viscous fluids. Viscous fluids flow out from the hole at the bottom of the container due to the gravity, and the fluid flow is captured with a smartphone fixed using a stand.

parameters in advance, and use these parameters to obtain extrinsic parameters. An image of a captured fluid video is given in Figure 6.4 (top left).

6.5.3 Fluid Data Extraction from Video

To compute the reference silhouette, I extract position data from the 2D example fluid videos. To this end, I use a standard background subtraction method based on Gaussian mixture modeling. Then, I separate the extracted silhouette, i.e., foreground from the background with a thresholding method. Finally, I perform closing and opening operations for the extracted foreground to remove some noisy estimates. I define the foreground as 1 and background as 0. The extracted silhouette is shown in Figure 6.4 (bottom left).

6.5.4 Screen Space Evaluation

To evaluate the objective function, I compute $\mathbf{g}$, silhouette of simulated fluids on the 2D screen space at each frame through the forward simulations. Since geometry of fluids is represented by a set of particles, I

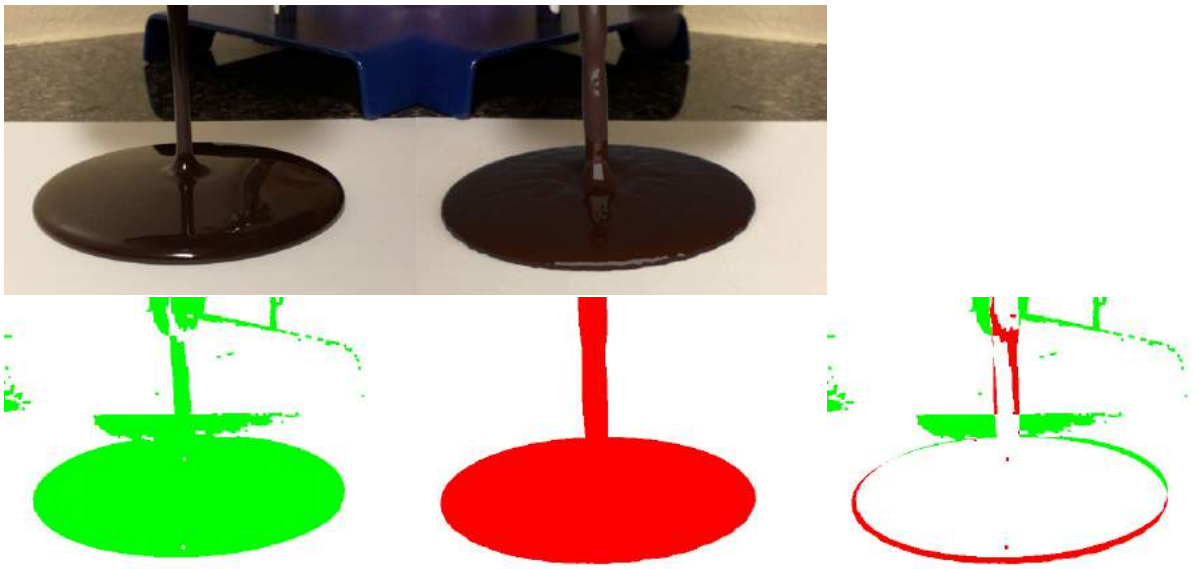


Figure 6.4: (Top) from left to right, example fluid video, and simulation result. (Bottom) Extracted silhouette from the example video, projections of fluid surfaces from the simulation, and the differences between the silhouette from the example data and simulation result.

first construct fluid surfaces to approximate the surfaces of the real fluid flows, and then project the surfaces onto the 2D screen space using the camera parameters which are used to capture the example videos.

To construct fluid surfaces, I take a standard approach. First, I generate implicit functions from the set of particles, construct surfaces using the marching cubes algorithm, and then perform several smoothing operations to better approximate the real fluid surfaces. In my work, I represent the surfaces with a set of triangles for the ease of projections onto the screen space.

Next, in the projection step, I form the silhouette of the fluid surfaces as a union of all the projected triangles on the screen space. To project each triangle, first, I independently project the three vertices of the triangle in the same way as the camera does. The projection operation can be written as

$$\mathbf{x} = \mathbf{KAX}, \tag{6.10}$$

where $\mathbf{X}$ and $\mathbf{x}$ denote the homogeneous coordinates of the vertex before and after projection, respectively, $\mathbf{K}$ and $\mathbf{A}$ intrinsic and extrinsic parameters, respectively, which can be computed with a camera calibration technique. After the projection of the three vertices, I can form a new triangle on the screen space. To compute $\mathbf{g}_t$, silhouette formed by a triangle t, I perform the inside/outside check for the center of each pixel, and I assign 1 to $\mathbf{g}_{t,i}$ if the center of pixel i is inside of the silhouette, otherwise 0. Finally, I assemble all the

silhouettes from the triangle to form the silhouette of the fluid surfaces, i.e.,

$$\mathbf{g} = \bigcup_t \mathbf{g}_t. \tag{6.11}$$

Figure 6.4 (bottom, middle) shows a computed silhouette from the simulation (top, middle), and silhouette difference is given in Figure 6.4 (bottom, right). After the projections of all the triangles, I can straightforwardly compute the objective function E.

6.5.5 Parameter Optimization

My objective function is formulated with example data captured in the real world, which generally include some noise, and involves multiple discontinuous operations, such as background subtraction and morphological operations, liquid domain computation and surface reconstruction from a set of particles, and projections of the 3D fluid surfaces onto the 2D screen space over multiple steps. Consequently, my objective function is discontinuous and nonlinear with many unacceptable local minima. Since evaluating analytical gradient is not practical for such discontinuous functions (McNamara et al., 2004), it is preferable to employ optimization methods based on sampling which can be used without evaluating the gradient analytically, as done in (Wang et al., 2015; Yang and Lin, 2016; Hu et al., 2017). In addition, sampling-based approaches can naturally satisfy the hard constraint for the physics in the constrained optimization problem (6.8) by performing forward simulations.

In my framework, I use a derivative-free optimizer, CMA-ES, which is known as robust to noise and efficient compared to other derivative-free optimization methods, such as particle swarm method and Nelder-Mead method. In the optimization, to enforce $0 \leq \eta$, I resample viscosity values if sampled viscosity values are smaller than 0.

While I have tested multiple gradient-based optimizers using finite difference approximations, such as L-BFGS, nonlinear conjugate gradient, and gradient descent with momentum, I found that these approaches almost always got stuck at suboptimal local minima because of the inaccurate estimates of the gradient for the noisy objective functions, and the computational cost for the convergence was higher than CMA-ES in most of my experiments.

6.6 Validations and Discussions

I implemented my framework in C++. For the parameter identification, I typically formulate the objective function with up to 100 video frames of high resolutions to make the optimization tractable (i.e., $N \leq 100$). I usually perform up to 80 iterations for CMA-ES optimization with an initial value between 0.0 and 3.0×1.0^2 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$ and standard deviation between 1.0×10^1 and 3.0×10^2 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. The overall computation time varies and depends on the video resolution, the number of video frames, the number of optimization iterations, the scale of fluid simulation, and the computational complexity of the (viscous) fluid solver. I used blender cycles renderer for Figure 6.4 and Figure 6.6 (first and second rows) and mitsuba renderer for the others.

I tested my framework in a wide range of scenarios. First, I validate the reliability of my algorithm with synthetic examples, and then I evaluate my framework with example videos captured in the real world.

6.6.1 Validation with Synthetic Videos

To test my framework, I generated several videos using my viscous fluid solver, and used the videos as input for my framework. The purpose of this experiment is to validate that my algorithm can identify viscosity parameters which are used to generate the synthetic videos, only with 2D data in the screen space.

I chose a scenario, where a viscous fluid flows from the hole at the bottom of a container, as shown in Figure 6.12 (top). In this scene, I tested with multiple viscosity values, $1.0 \times 10^0, 3.0 \times 10^0, 1.0 \times 10^1, 3.0 \times 10^1, 1.0 \times 10^2$, and 3.0×10^2 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. The simulation parameters and identification results are summarized in Table 6.1, and I note that this experiment covers a sufficiently wide range of Reynolds numbers for viscous fluids and thus fluid behaviors. A plot for the objective function is given in Figure 6.5. The second row of Figure 6.12 demonstrates simulation results in the same scenario with the identified parameters, and in general, visual differences between the reference and the simulated videos are indiscernible.

One advantage of my framework with iterative inversion using the full 3D simulation is that I can infer hidden physical variables which are not available from the video data, e.g., velocity of fluid flows and pressure distributions. Figure 6.12 visualizes the pressure and velocity distributions for the input example (third and fifth rows) and simulation results with the identified parameters (fourth and sixth rows). Similar to the comparison with the surface rendering, differences for the pressure distributions and velocity fields between real and virtual fluids are generally indiscernible.

Table 6.1: Viscosity parameter identification results with synthetic videos. $\hat{\eta}$ denotes reference fluid viscosity ($\mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$), Re Reynolds number, η identified viscosity value ($\mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$), ϵ_η, ϵ_v, and ϵ_p relative errors for the viscosity (%), pressure (%), and velocity (%), respectively. The error for the viscosity is relatively small and up to around 5%.

$\hat{\eta}$	Re	η	ϵ_η	ϵ_p	ϵ_v
1.0×10^0	1.25×10^2	1.06×10^0	5.85	14.1	1.21
3.0×10^0	3.67×10^1	2.94×10^0	1.93	6.60	0.57
1.0×10^1	9.50×10^0	0.98×10^1	1.52	8.28	0.56
3.0×10^1	2.67×10^0	3.02×10^1	0.80	4.34	0.32
1.0×10^2	6.01×10^{-1}	1.01×10^2	1.35	3.34	0.74
3.0×10^2	1.52×10^{-1}	3.10×10^2	3.23	15.6	0.87

In these experiments, I used the same solver for synthetic example generation and parameter identification, and thus resulting fluid behaviors are same if the same viscosity values are used. However, I note that the example data include only rendered fluid surfaces generated by 3D fluid simulations, i.e., projected onto the 2D screen space (losing full 3D information which can be perfectly matched), and positional data are extracted with image processing algorithms, introducing some errors. Consequently, it is not guaranteed that my algorithm finds the ground truth, and the value of the objective function is 0. Nonetheless, my framework can identify viscosity parameters with *up to around 5%* relative errors, only with the 2D information, and the inferred pressure and velocity values are within 20% and 2% of relative errors, respectively (see Table 6.1). The plot in Figure 6.5 also demonstrates that the good local minimum is very close to the ground truth viscosity values while the objective function increases as viscosity values deviate from the ground truth.

6.6.2 Identification with Real World Captured Data

I also tested my framework with example videos captured from real world fluid phenomena. In this study, I experimented with caramel, red hand soap, chocolate syrup, purple body soap, blue body soap, and honey. To perform the parameter optimization, I setup the simulation scenarios as close as possible to the scene for the real experiments. The captured videos and simulation results with identified viscosity parameters are shown in Figure 6.6. A plot for the objective function with different viscosity values is given in Figure 6.7, and a plot for the convergence behaviors is given in Figure 6.8. Statistics and performance are summarized in Table 6.2.

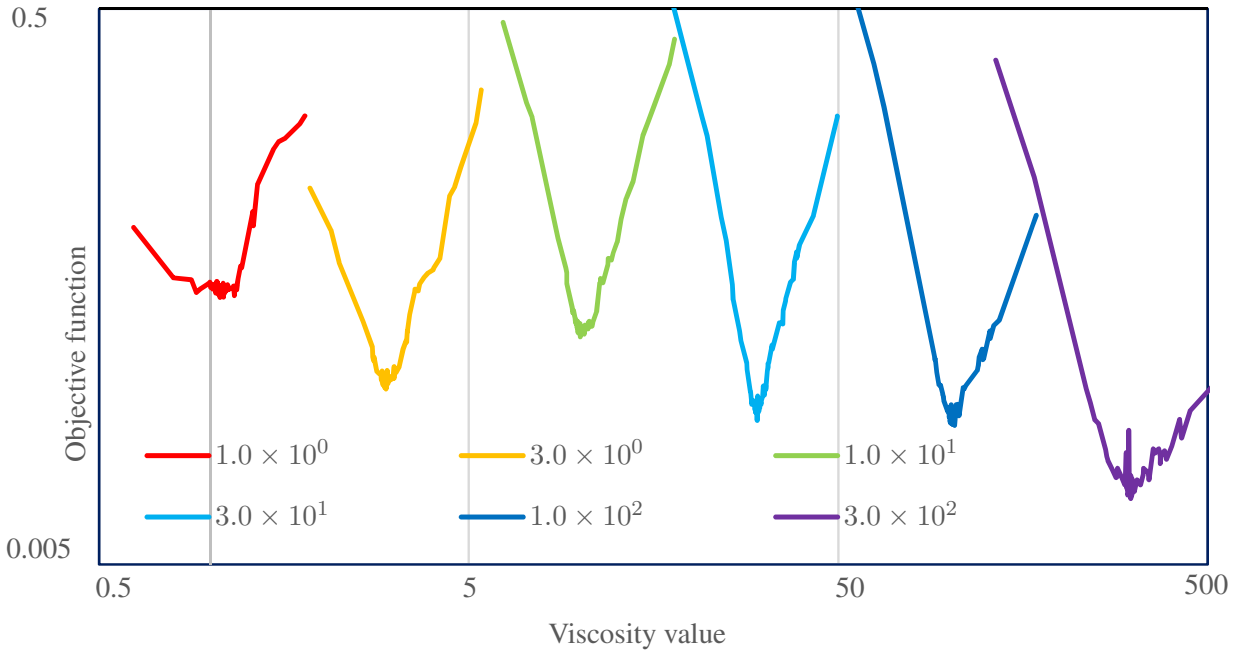


Figure 6.5: Plots of the objective functions with different viscosity values for Figures 6.12. Good local minima are located close to the ground truth. .

Since the ground truth of viscosity parameters are not available for viscous fluids in my examples, it is not possible to validate the accuracy of the identified parameters. However, Figure 6.6 demonstrates that the behaviors of the simulated viscous materials with the identified parameters are visually in close agreement with the fluids in the example videos. In addition, I note that the range of viscosity values for honey is known as between 2.0 and 10.0 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$, and my identified viscosity value for the honey is 7.86 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$ and is within the range, which further validates the reliability of my framework. Furthermore, I note that my framework can identify the viscosity parameters for fluids exhibiting the coiling behaviors, reproducing the buckling phenomena for the blue body soap and honey.

Similar to the case for the synthetic videos, one advantage of the iterative inversion using the 3D simulation is that I can estimate hidden variables for the real fluid flows, e.g., pressure and velocity profiles (which are not available in the example videos), as shown in Figure 6.6 (third and fourth rows). To validate the accuracy of the simulation with the identified parameters, I compare the flow speed of the fluids on the ground, which can be estimated from the example videos. Results are summarized in Table 6.2, and the average relative errors are up to around 10% in my experiments.

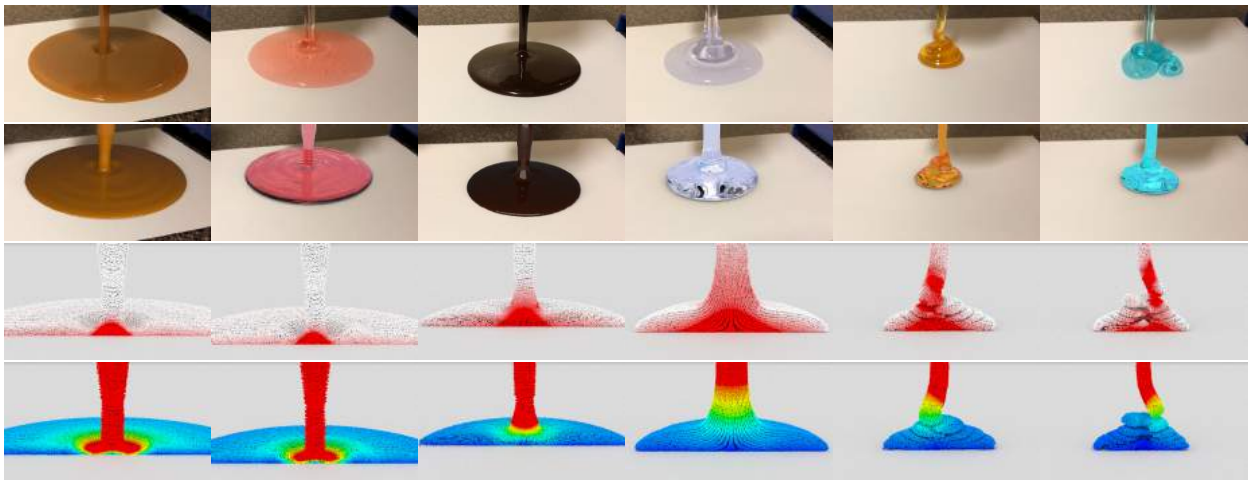


Figure 6.6: Parameter identification results with example videos from the real-world fluid phenomena. (Top) from left to right, caramel, red hand soap, chocolate syrup, purple body soap, honey, and blue body soap. (Bottom) simulation results with identified viscosity parameters, $\eta = 0.12, 0.31, 0.63, 2.05, 7.83$, and $8.81\ \mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$.

Table 6.2: Viscosity parameter identification results with example videos captured from real world fluid flows. Re denotes Reynolds number, η identified viscosity value ($\mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$), t average time in minutes for each iteration, T total time in hours for the parameter identification, and $\hat{v}$ and v (cm/s) average flow speed of the fluids estimated from the video and computed from the simulation, respectively.

Materials	Re	η	t	T	$\hat{v}$	v
Caramel	3.16×10^{1}	0.19	26.3	10.8	6.1	6.7
Red hand soap	2.72×10^{1}	0.22	14.3	13.9	5.2	5.8
Chocolate syrup	4.80×10^{0}	1.25	15.2	7.7	4.5	5.3
Purple body soap	6.33×10^{-1}	4.74	19.4	11.7	1.7	1.5
Blue liquid soap	5.15×10^{-1}	5.82	8.6	8.0	1.1	1.3
Honey	2.67×10^{-1}	7.86	7.5	9.0	0.6	0.6

6.6.3 Real-to-Virtual Parameter Transfer

The identified viscosity parameters can be used in novel scenarios. Figure 6.9 demonstrates a chocolate coating for a cake with the identified viscosity parameter for the ganache. Figure 6.10 shows a honey pouring onto a honey dipper with the identified parameter of the honey. Figure 6.11 demonstrates a pouring of magenta hand soap, light-lavender body soap, and aqua-green shampoo onto a hand with the identified parameters. Note that the differences in fluid behaviors in the example videos are sufficiently reflected in this scene, generating distinct fluid flows. Thus, I believe that it is undesirable to randomly choose viscosity parameters from the range of typical viscosity values for soap materials, even if such data are available.

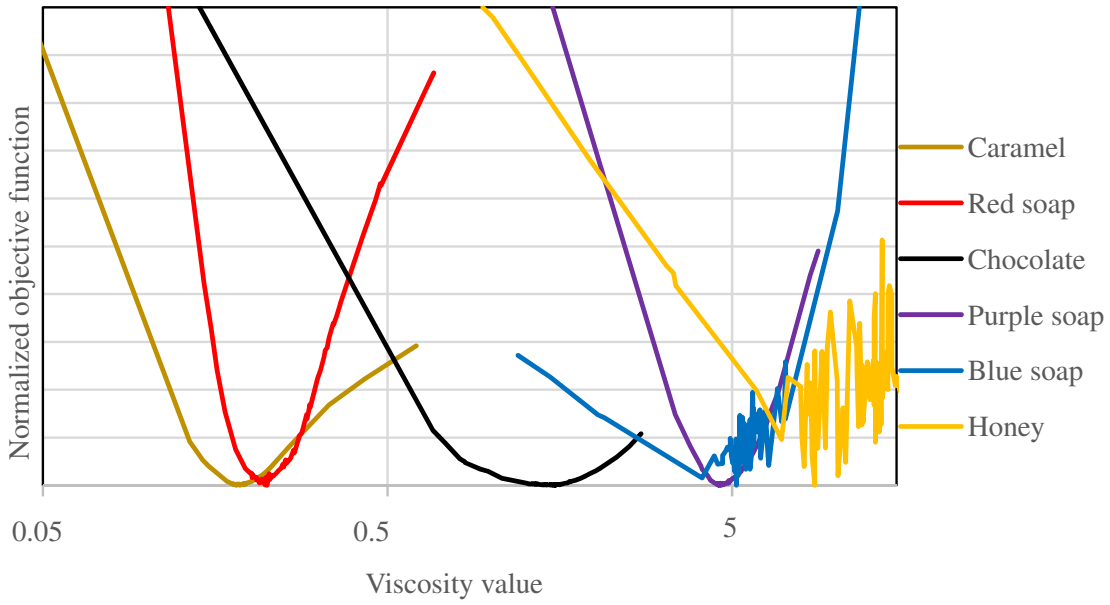


Figure 6.7: Plots of the objective functions with different viscosity values for Figure 6.6.

Figure 6.1 (right) demonstrates a scene with simulated donuts covered by chocolate syrup, caramel, and honey with the identified parameters. In this scene, I also clearly observe that caramel, chocolate syrup, and honey behave very differently according to their material properties.

6.6.4 Discussions

My framework can identify material parameters effectively as demonstrated. However, it is not guaranteed that the resulting parameters are close to the measured parameters unless the experiments are conducted under relatively ideal, controlled conditions. There are some factors for this discrepancy. First, fluid simulation is a numerical approximation of the complex fluid flows with a relatively simplified model derived based on various assumptions (e.g., no slip boundary condition and uniformly distributed fluid particles), which might not hold in some cases. In addition, while my focus is on purely Newtonian fluids, some real-world materials exhibit non-Newtonian properties as well, and thus simulation results would deviate from the real fluid behaviors. Given the relatively coarse simulation resolution, it is not possible to accurately capture the small scale details of fluids and solid boundaries, and their resulting influence to the simulation (e.g., neglected boundary details and strong surface tension due to thin fluid sheets).

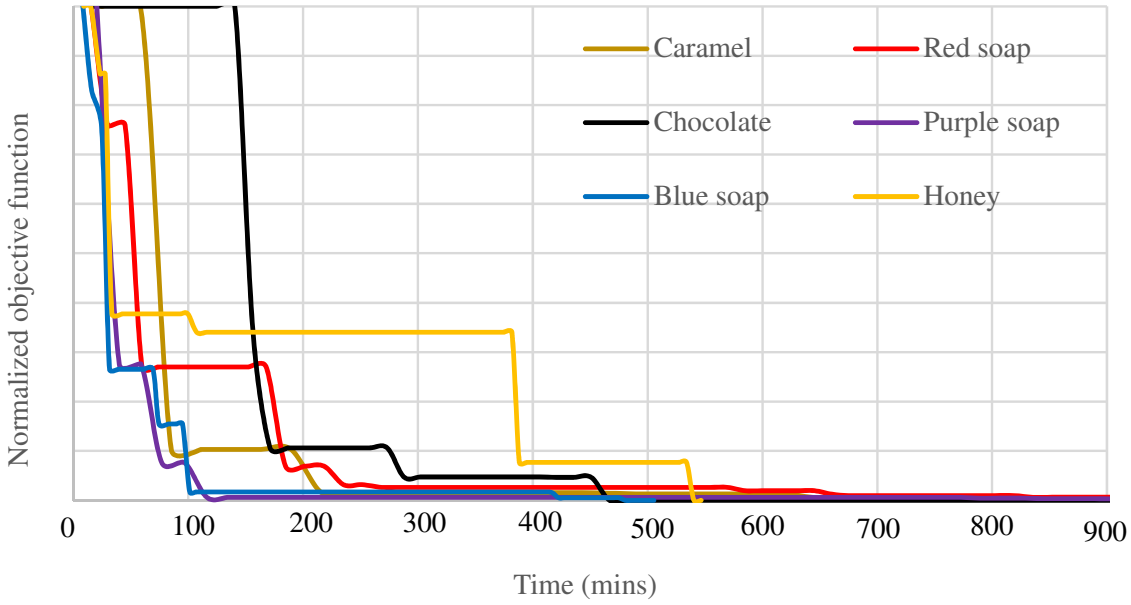


Figure 6.8: Convergence plot for the parameter identification in Figure 6.6.

6.7 Conclusions and Future Work

I proposed perhaps the first parameter identification framework to automate parameter-tuning for fluid simulation with example data captured from real-world fluid phenomena. My framework takes example fluid videos as a reference and minimizes the differences between the reference and simulated fluids (using my solver) to identify material parameters. For the difference measurement with example videos, I presented a screen space evaluation method, which compares the reference and simulation results on the 2D screen space, avoiding erroneous and ambiguous 3D reconstruction of fluid data. I validated my parameter identification framework with a range of synthetic and real-world data and demonstrated that identified material parameters can be effectively used to infer hidden physical variables of real fluids and to simulate viscous fluids in novel scenarios, generating fluid behaviors visually consistent to the example data.

There are several promising future research directions. In the real world, there are many different types of fluid materials, such as non-Newtonian fluids, which require more complex constitutive laws to simulate . In addition, material parameters for fluids can vary, e.g., due to heat and stress. It would be interesting to develop parameter identification framework that can take into account more sophisticated physics and property change.

Figure 6.9: Simulated chocolate ganache poured onto a cake with the identified viscosity parameter $\eta = 1.25\ \mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$.

In general, it is difficult to obtain meaningful and reliable fluid data with simple computer vision techniques, such as background subtraction, from normal videos available in public. Thus, it would be necessary to explore some descriptors for fluids which can be reliably used for the difference measurements. I would also like to investigate advanced computer vision techniques and deep learning, to extract fluid information. Along this direction, I believe that learning-based approaches for video analysis and processing are promising.

Acknowledgements

This work is supported in part by National Science Foundation and Elizabeth Stevinson Iribe Chair Professorship. We would like to thank Christopher Batty and Ryoichi Ando for discussions on the viscous fluid solver and anonymous reviewers for their valuable suggestions and comments.

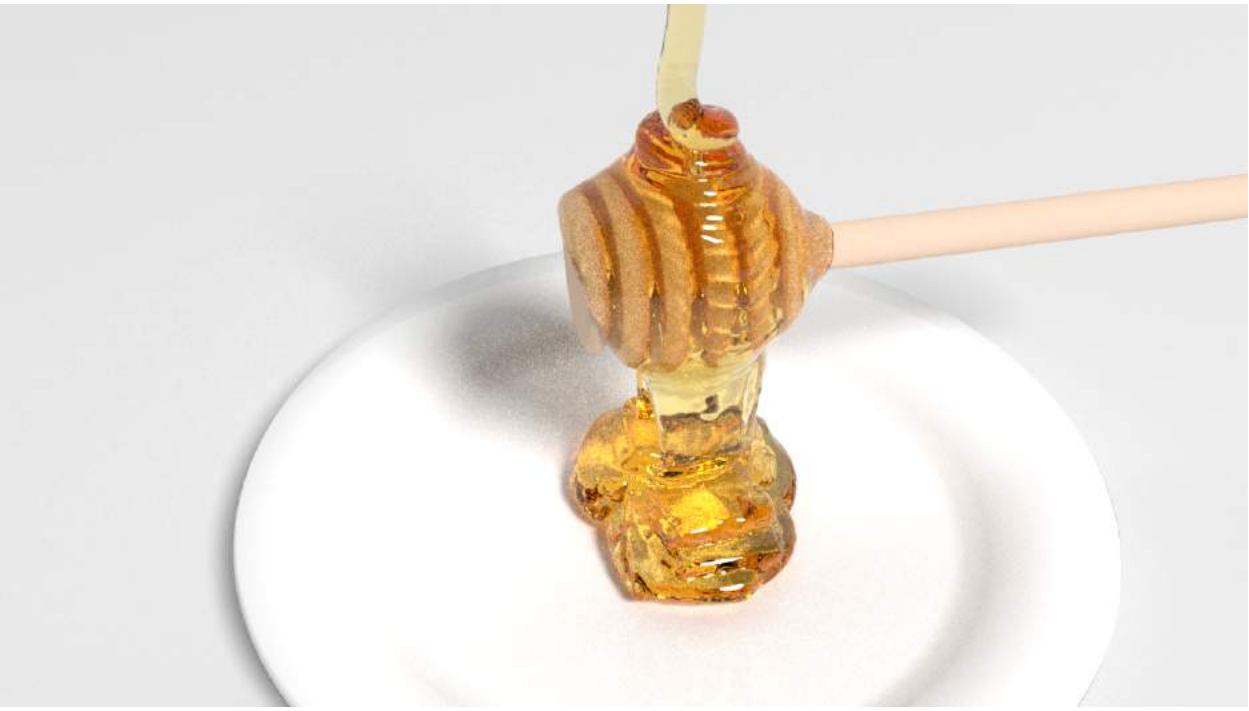


Figure 6.10: Virtual honey dripped onto a honey dipper with the identified viscosity parameter $\eta = 7.86\ \mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$.

Figure 6.11: From left to right, magenta hand soap, light-lavender body soap, and aqua-green shampoo poured onto a hand with the identified viscosity parameters $\eta = 0.22, 4.74$, and $5.82\ \mathrm{kg}/(\mathrm{s}\cdot\mathrm{m})$, respectively.

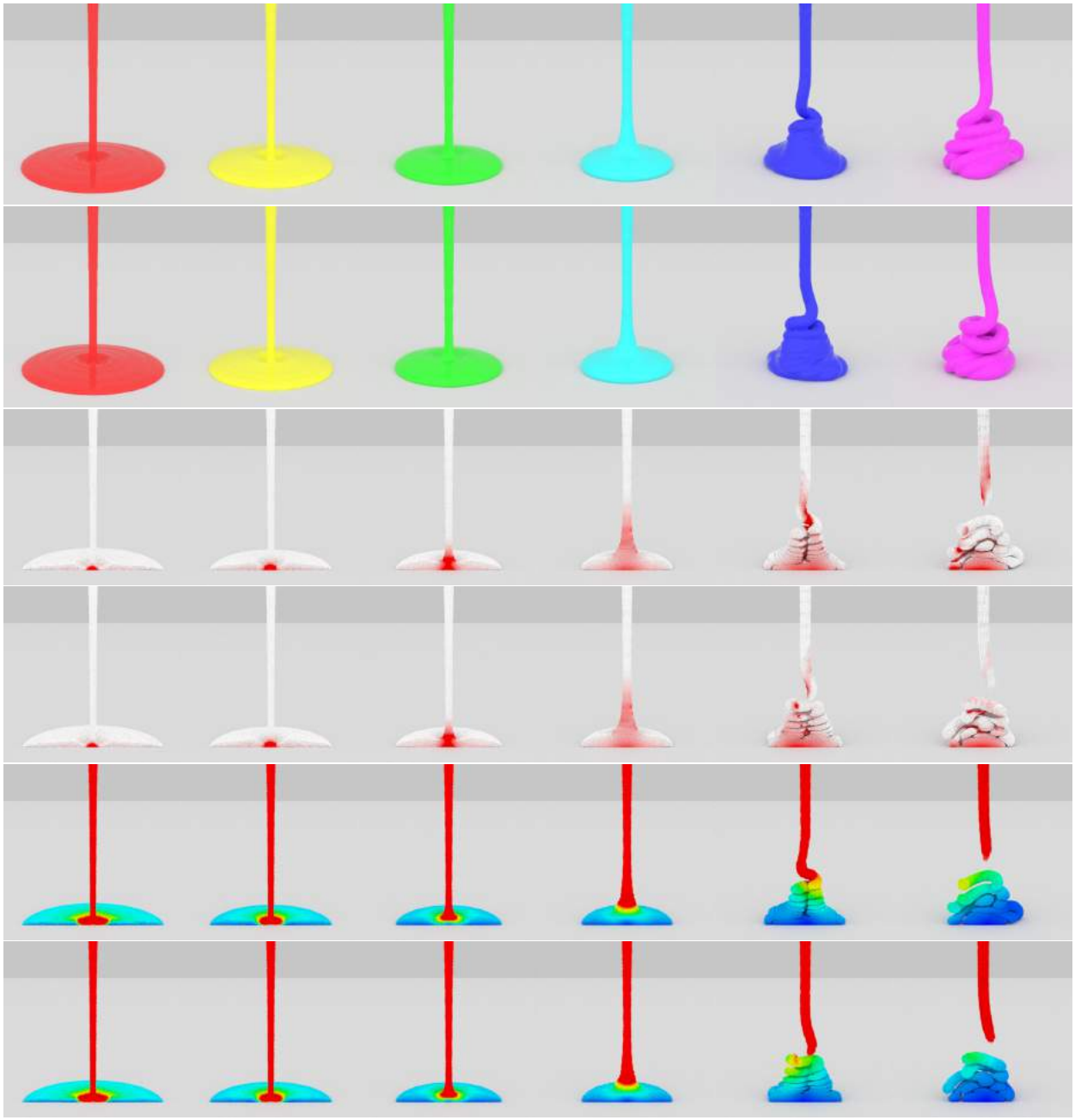


Figure 6.12: Validation results with synthetic videos for the scenario of flowing fluids. (First row) from left to right, simulated video as input, with viscosity parameters $\eta = 1.0\times10^0, 3.0\times10^0, 1.0\times10^1, 3.0\times10^1, 1.0\times10^2$, and 3.0×10^2 kg/(s · m). (Second row) recovered results using my framework with identified viscosity parameters, $\eta = 1.06\times10^0, 2.94\times10^0, 0.98\times10^1, 3.02\times10^1, 1.01\times10^2$, and 3.10×10^2 kg/(s · m). The relative errors are $5.85\%, 1.93\%, 1.52\%, 0.80\%, 1.35\%$, and 3.23%, respectively. Cutaway particle visualization for pressure profiles for the input (third row) and simulation with the identified parameters (fourth row). Cutaway particle visualization using rainbow colors for velocity profiles as the input (fifth row) and simulation with the identified parameters (sixth row).

CHAPTER 7: Conclusion

In this dissertation, I have proposed various methods for simulating incompressible viscous liquids and an optimization framework for the video-guided parameter transfer. These proposed methods enabled robust, efficient, and scalable simulations of particle-based fluids, capturing of characteristic rotational behaviors of viscous fluids and more accurate two-way fluid-solid interactions, and transfer of viscosity parameters from the real world to the virtual environment. I have demonstrated the effectiveness and capability of the proposed approaches in a wide variety of scenarios. The proposed methods and framework can be used not only for computer graphics applications but also for engineering applications.

7.1 Summary of Results

To efficiently simulate incompressible fluids, I first proposed a particle-based solver based on incompressible SPH. The method includes three main contributions: kernel blending, free surface boundary handling, and solid boundary handling. To consistently blend density computed with spiky and smooth kernels, I normalized the density computations for both kernels, and then blended the computed densities. This scheme improved the robustness of the fluid solver and allowed for taking larger time steps with a negligible computational cost. For the free surface boundary handling, I first analyzed possible configurations and the cause of the failure in the pressure solve. Based on the analysis, then, I appropriately classify particle roles in the pressure solve. Consequently, the free surface handling method also improved the robustness without introducing artifacts, as compared to previous techniques. For the solid boundary handling, I also analyzed possible particle configurations and detected the failure cases. To address the failure cases, I introduced a new regularizer term to ensure the solvability of the linear system regardless of the particle configurations. This scheme further improved the robustness and accuracy in the pressure solve. In addition, the solid boundary handling method enabled us to simulate objects floating in the air. By combining these contributions, my incompressible fluid solver outperformed one of the state-of-the-art particle-based solver, IISPH by a factor of 3.78 (Chapter 2, (Takahashi et al., 2016)).

Simulating incompressible liquids can be computationally expensive. This becomes more noticeable when we simulate fluids at high-resolution since the computational cost for solving incompressibility constraints increases superlinearly. To address this issue, I proposed a geometric multilevel solver for efficiently solving linear systems arising from particle-based methods. To apply this method to particle systems, I constructed the hierarchy of the simulation particles and grids, and established the correspondence between solutions at the particle and grid levels. I also proposed a coarsening scheme, which takes simulation elements into account and can ensure the solvability of the linear systems at coarser levels. In addition, I proposed a new solid boundary handling method for solving a pressure Poisson equation in a unified manner. I have demonstrated that the computational cost of my multilevel solver scales nearly linearly with respect to the number of particles, unlike previous particle-based methods that use Jacobi or CG solvers, outperforming the state-of-the-art IISPH solver up to 10x, and my solver can be used to simulate various complex scenarios, including moving solid boundaries and two-way fluid-solid interactions (Chapter 3, (Takahashi and Lin, 2016)).

Unlike inviscid fluids, viscous fluids exhibit characteristic behaviors, such as coiling and buckling phenomena, and specialized techniques have been required to capture such behaviors of viscous fluids. I have proposed a stable and efficient particle-based solver for simulating highly viscous fluids to capture rotational behaviors of fluids and handle spatially varying viscosity values. In contrast to previous methods that use explicit viscosity integration, I employed an implicit integration and formulated the implicit viscosity integration as a minimization problem based on the variational principle, which naturally enforces the boundary conditions. The minimization is quadratic with respect to particle velocities, and thus the minimization problem can be handled by solving a linear system with a sparse positive definite matrix. To efficiently solve the resulting linear system for the implicit viscosity integration, I proposed a coefficient extraction method, which assembles a linear system accessing neighbor particles' neighbor particles. The method enabled the use of larger time steps and higher viscosity values. By taking advantage of our implicit formulation and coefficient extraction method, I achieved an accelerated performance by a factor of 3.4 compared to previous methods that use explicit integration (Chapter 4, (Takahashi et al., 2015)).

While the recent research due to the high demand for numerous applications significantly advanced viscous fluid simulation approaches, few research has been conducted to simulate how viscous fluids interact with solid objects. Thus, I have proposed a grid-based fluid solver for simulating viscous materials and their interactions with solid objects. My method formulates the implicit viscosity integration as a minimization

problem for robustness and efficiency. To handle the interplay between fluids and solid objects with viscosity forces, I also formulate the two-way fluid-solid coupling as a unified minimization problem based on the variational principle, which naturally enforces the boundary conditions. My formulation leads to a symmetric positive definite linear system with a sparse matrix regardless of the monolithically coupled solid objects. To improve the accuracy, I also propose a geometrically-consistent volume estimation method that can more accurately compute the volume fractions for the minimization, taking into account the sub-grid details of free surfaces and solid boundaries. Additionally, I presented a position-correction method using density constraints to enforce the uniform distributions of fluid particles, preventing the loss of fluid volumes. I evaluated the accuracy of the proposed method by comparing the results with the analytical solutions, and improved the accuracy by several orders of magnitude, compared to previous techniques. I have demonstrated the effectiveness of my method in a wide range of viscous fluid scenarios (Chapter 5, (Takahashi and Lin, 2019a)).

In physically-based simulation, it is essential to choose appropriate material parameters to generate desirable simulation results. In many cases, however, choosing material parameters is very challenging, and often tedious trial-and- error parameter tuning steps are inevitable. Thus, I proposed a real-to-virtual parameter transfer framework for helping the parameter identification of viscous fluids with example video data captured from real-world phenomena. My method first extracts positional data of fluids and then uses the extracted data as a reference to identify the viscosity parameters, combining forward viscous fluid simulations and parameter optimization in an iterative process. I evaluated my method with a range of synthetic and real-world example data, and demonstrated that my method can identify the hidden physical variables and viscosity parameters. This set of recovered physical variables and parameters can then be effectively used in novel scenarios to generate viscous fluid behaviors visually consistent with the example videos (Chapter 6, (Takahashi and Lin, 2019b)).

7.2 Limitations

I discussed specific limitations for each of my proposed methods and framework in their corresponding chapters. In this chapter, I discuss the limitations on my general approach. Fluid simulation is a powerful and essential tool to simulate fluid behaviors because we can efficiently and conveniently predict or visualize the dynamics of fluids without repetitive manual processing and fluid capturing in the real-world. However,

fluid simulation is one way to approximate the behaviors of real fluids based on the Navier-Stokes equations, and thus there exist some differences between behaviors of real fluids and simulated fluids. In addition, the Navier-Stokes equations are one model simplified to make the simulation tractable on computers. As such, even though we accurately solve the Navier-Stokes equations, the discrepancy between real and simulated fluids may still exist. In practice, there are so many factors for the discrepancy, including physical model differences, resolution limits, geometry approximation, simplification for boundaries. Because of these and many other unknown factors, the results of the simulations are not necessarily closer to the real-world counterparts even though the accuracy of the simulation is increased with the proposed methods, closely following the solutions for the Navier-Stokes equations. To more closely approximate the real world fluid phenomena or to generate some user-desirable fluid effects, it is necessary to establish a mathematical model to describe the desired fluid dynamics.

7.3 Future Work

While the proposed methods improved the efficiency and robustness for the viscous fluid simulation, and the parameter transfer framework facilitated the parameter tuning, still there are many open challenges. Here, I list some of challenges.

In fluid simulation, handling boundary conditions have been investigated over many years since it is generally not possible to fully resolve the real boundaries in the simulation, and we are required to approximate the boundary conditions as accurate as possible. However, because of the approximation, it is challenging to generate fluid effects closely matching the real world counterparts while maintaining numerical stability issues. In Chapter 2, I have proposed a method for handling free surfaces and solid boundaries, and this improved the efficiency and robustness. However, it is not clear whether these boundary handling schemes are appropriate in terms of physical fidelity. In Chapters 4 and 5, I presented viscous fluid simulation methods assuming the no-slip boundary conditions for the solid boundary because it is empirically known that the no-slip boundary condition leads to better behaviors for viscous fluids. However, this treatment is inappropriate for less viscous fluids, and such mixture of highly viscous fluids and inviscid fluids cannot be appropriately handled. While one reason for this problem is due to the numerical approximation for simulation, another reason is that, still how fluids behave on the solid boundaries is not well understood in terms of the mechanism. As such, we would need to explore how fluid behaves on the solid boundaries, and

then how to approximate the real fluid behaviors on the boundary in simulation. While I focused on boundary conditions related to my works in this dissertation, how to handle boundary conditions is an essential problem, which can arise in many different simulation and applications.

The expensive cost for the fluid simulation has been also an major issue. While I focused on the algorithmic improvement to address the issue in this dissertation, there are other ways to tackle this issue. One of them is to exploit parallel computations. While most of the presented algorithms are already parallelized with multi-core CPUs and shared memory, I expect that using many-core CPUs with distributed memory would further improve the performance, and similarly GPU would significantly accelerate the fluid simulation. The use of these parallel computation architectures would enable much more efficient simulations.

To address the parameter tuning problem, I proposed a parameter identification framework for viscous fluids in Chapter 6. While I focused on the viscosity parameter for fluids, the framework can be extended or adapted to identify different parameters for fluids, such as surface tension, viscoelasticity, and plasiticity. Furthermore, the framework could also be extended not only for fluids but other materials, e.g., to rigid bodies, deformable objects, and granular materials. I hope that the presented framework for parameter identification with real-world video data opens a door for new directions.

APPENDIX A: ALGORITHMS FOR MULTILEVEL SOLVER

For implementation of my method, I give algorithms for my incompressible SPH solver (Algorithm 5), the PPE solve (Algorithm 6), and V-cycle (Algorithm 7).

Algorithm 5 Incompressible SPH Solver

for all particle i **do**
  find neighbor fluid and solid particles
for all fluid particle i **do**
  compute density ρ_i
for all fluid particle i **do**
  predict intermediate velocity $\mathbf{u}_i^*$
for all fluid particle i **do**
  predict intermediate density ρ_i^*
Solve the PPE (Algorithm 6)
for all particle i **do**
  compute pressure force $\mathbf{F}_i^{\mathrm{p}}$
for all particle i **do**
  integrate velocity $\mathbf{u}_i$ and position $\mathbf{x}_i$

Algorithm 6 PPE Solve

for all fluid particle i **do**
  if i is Dirichlet or isolated particle **then**
    $p_i = 0$
  else
    for all neighbor solid particle s **do**
      compute $\frac{1}{\delta_s}\nabla W_{is}$
    compute $\alpha_i = \frac{\rho_0^2}{\rho_i^2}\| \sum_s \frac{1}{\delta_s}\nabla W_{is}\|^2$ and $b_i = \frac{\rho_i^* - \rho_0}{\Delta t^2}$
    if i is separated particle **then**
      $p_i = \max(0, \frac{b_i}{\alpha_i})$
    else if i is Poisson particle **then**
      for all neighbor fluid particle j **do**
        compute a_{ij}
      Assemble the matrix and source term (right hand side)
Solve the PPE with my MGCG solver
for all Poisson particle i **do**
  $p_i = \max(0, p_i)$

Algorithm 7 V_cycle(l)

1: //$\mathbf{r}$: residual
2: //k^{pre} and k^{post}: the number of pre- and post-smoothing.
3: **if** $l == 0$ **then**
4: Solve $\mathbf{A}^0\mathbf{p}^0 = \mathbf{b}^0$
5: **else**
6: **for** $k = 1$ to k^{pre} **do**
7: $\mathrm{PreSmooth}(\mathbf{A}^l, \mathbf{b}^l, \mathbf{p}^l)$
8: $\mathbf{r}^l = \mathbf{b}^l - \mathbf{A}^l\mathbf{p}^l$
9: $\mathbf{b}^{l-1} = \mathrm{Restrict}(\mathbf{r}^l)$
10: $\mathbf{p}^{l-1} = 0$
11: $\mathrm{V\_cycle}(l-1)$
12: $\mathbf{p}^l = \mathbf{p}^l + \mathrm{Interporate}(\mathbf{p}^{l-1})$
13: **for** $k = 1$ to k^{post} **do**
14: $\mathrm{PostSmooth}(\mathbf{A}^l, \mathbf{b}^l, \mathbf{p}^l)$

APPENDIX B: DETAILS OF COEFFICIENT EXTRACTION

Due to complexity of extracting coefficients and similarity of computing coefficients, I divide them into three groups: G_i ($c_{u_iu_i}, c_{v_iu_i}$, and $c_{w_iu_i}$), G_j ($c_{u_ju_i}, c_{v_ju_i}$, and $c_{w_ju_i}$), and G_k ($c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$).

By scanning particle j once from particle i, I can easily compute and extract coefficients in G_i.

Since computing coefficients in G_j requires $\mathbf{a}_{ij}$ and ω_{ij} (summation of ∇W_{ij} over particle j), and ∇W_{ij} and α_{jk} (summation of $V_k \nabla W_{jk}$ over particle k), I need to compute ω_{ij} and α_{ij} (I can access α_{jk} from particle j if α_{ij} is computed in a previous loop) in advance using another loop unlike the case of G_i. Therefore, I first compute ω_{ij} and α_{ij} in the first loop and then use them to compute and extract coefficients in G_j in the second loop. Computed ω_{ij} and α_{ij} can also be used to obtain coefficients in G_i.

To compute coefficients in G_k, I need to access particle k from particle i. However, since particle k is a neighbor of particle j, I cannot directly access particle k from particle i. Hence, I access particle k via particle j. In that case, however, I cannot take a sum of quantities computed between particles i and j at particle k (another scan for particle j at particle k leads to quadruple loops, which are costly and make coefficient extractions more complex). Therefore, I decompose coefficients $c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$ without taking a sum over particle j and separately extract coefficients at particle k. Assuming that I am accessing particle k^* via particle j^*, I can write a part of coefficients $c_{u_ku_i}|_{j^*k^*}, c_{v_ku_i}|_{j^*k^*}$, and $c_{w_ku_i}|_{j^*k^*}$ for $c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$ from Eqs. (4.10), (4.11), and (4.12) as

$$
\begin{aligned}
c_{u_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \\
(2\nabla W_{ij^*,x} a_{j^*k^*,x} &+ \nabla W_{ij^*,y} a_{j^*k^*,y} + \nabla W_{ij^*,z} a_{j^*k^*,z}), \\
c_{v_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \nabla W_{ij^*,y} a_{j^*k^*,x}, \\
c_{w_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \nabla W_{ij^*,z} a_{j^*k^*,x}.
\end{aligned}
$$

By adding the part of coefficients above to the matrix at particle k while also scanning particle j from particle i, I can extract all coefficients of the matrix.

Adding coefficients to the matrix separately does work. However, this approach is inefficient because I generally use structures specialized for a sparse matrix, e.g., CSR and Coordinate list, and adding values to such structures frequently is costly. To avoid this, I use auxiliary storage, which is associated with particle i,

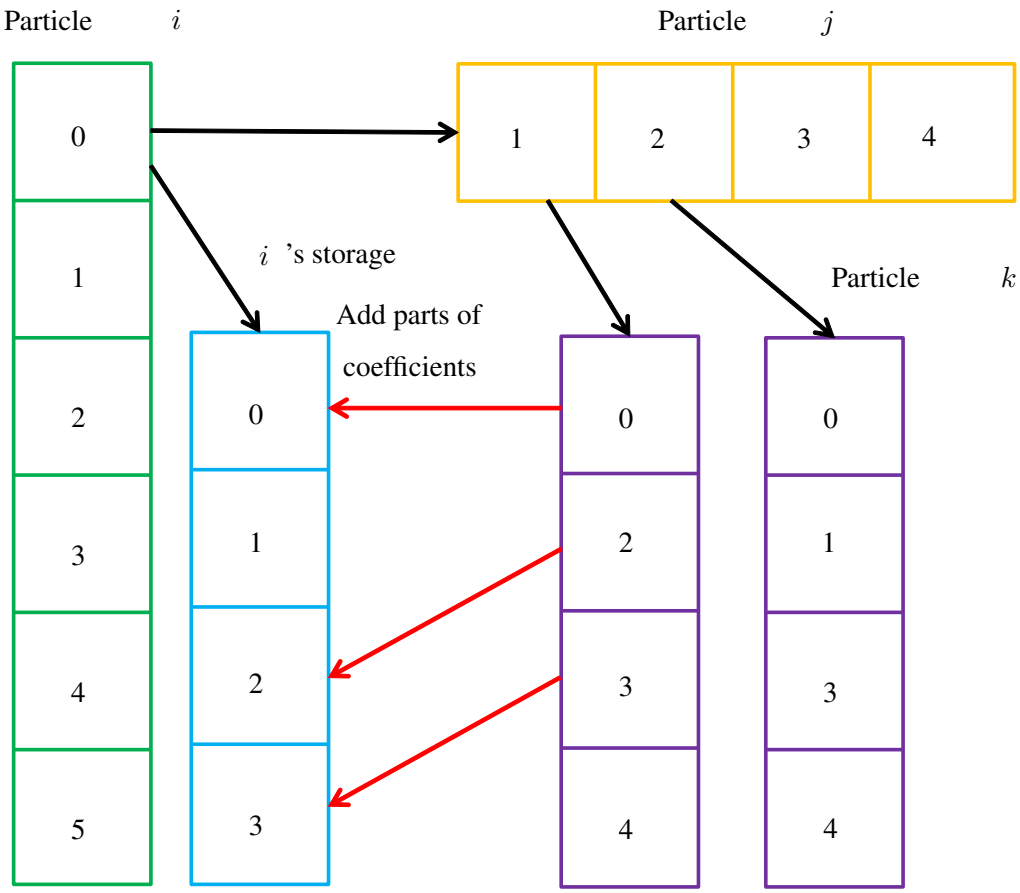


Figure B.1: Illustration of computational flow for extracting coefficients in G_k. Particle k is accessed from particle i via particle j. At particle k, parts of coefficients in G_k are added to a cell in particle i's storage, whose id matches k's id, as shown by red arrows. Black arrows represent accessible particles and storage.

to preserve coefficients from particle k to i with k's id. I add parts of coefficients to i's storage at particle k, grouping them based on k's id to minimize the number of adding coefficients to the matrix. Then, after scans over particles j and k from particle i are finished, I add coefficients in G_k to the matrix using the storage. In my experiments, minimizing the number of access to the matrix using the auxiliary storage accelerated my coefficient extractions by a factor of 4.5 as compared to adding coefficients to the matrix directly at particle k. A flow for coefficient extractions for G_k is illustrated in Figure B.1.

I also perform similar procedures explained above to extract coefficients $c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_kv_i}, c_{v_kv_i}$, and $c_{w_kv_i}$ for v_i and $c_{u_iw_i}, c_{v_iw_i}, c_{w_iw_i}, c_{u_jw_i}, c_{v_jw_i}, c_{w_jw_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ for w_i. I show my algorithm for coefficient extraction in Algorithm 8.

Algorithm 8 Algorithm for coefficient extraction

initialize a matrix
for all fluid particle i **do**
  compute $\hat{\mu}_i, \omega_{ij}$ and α_{ij}
compute $\hat{m}$
for all fluid particle i **do**
  initialize storage for u_k, v_k, and w_k
  add $c_{u_iu_i}, c_{v_iu_i}, c_{w_iu_i}, c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_iw_i}, c_{v_iw_i}$, and $c_{w_iw_i}$ to the matrix
  for all fluid particle j **do**
    compute ∇W_{ij} and $\mathbf{a}_{ij}$
    add $c_{u_ju_i}, c_{v_ju_i}, c_{w_ju_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_jw_i}, c_{v_jw_i}$, and $c_{w_jw_i}$ to the matrix
    for all fluid particle k **do**
      compute $\mathbf{a}_{jk}$
      add $c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ to the i's storage with k's id
  for all i's storage **do**
    add $c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ to the matrix using the storage

APPENDIX C: DERIVATION OF IMPLICIT FORMULATION

I derive our implicit formulation for the full form of viscosity using SPH. I have the following equations for the viscosity term.

$$\mathbf{u}_i = \mathbf{u}_i^* + \frac{\Delta t}{\rho_i}\nabla \cdot \mathbf{s}_i, \tag{C.1}$$

$$\mathbf{s}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T\right) \tag{C.2}$$

$$\nabla \cdot \mathbf{s}_i = m\rho_i \sum_j \left(\frac{\mathbf{s}_i}{\rho_i^2} + \frac{\mathbf{s}_j}{\rho_j^2}\right) \nabla W_{ij} \tag{C.3}$$

$$\mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left(\frac{\mathbf{s}_i}{\rho_i^2} + \frac{\mathbf{s}_j}{\rho_j^2}\right) \nabla W_{ij} \tag{C.4}$$

$$\mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left(\frac{\mu_i}{\rho_i^2}\left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T\right) + \frac{\mu_j}{\rho_j^2}\left(\nabla \mathbf{u}_j + (\nabla \mathbf{u}_j)^T\right)\right) \nabla W_{ij} \tag{C.5}$$

$$\nabla \mathbf{u}_i = \sum_j V_j (\mathbf{u}_j - \mathbf{u}_i) \nabla W_{ij}^T \tag{C.6}$$

$$= \sum_j V_j \begin{bmatrix} u_j - u_i \\ v_j - v_i \\ w_j - w_i \end{bmatrix} \begin{bmatrix} \nabla W_{ij,x} & \nabla W_{ij,y} & \nabla W_{ij,z} \end{bmatrix} \tag{C.7}$$

$$= \sum_j V_j \begin{bmatrix} (u_j - u_i)\nabla W_{ij,x} & (u_j - u_i)\nabla W_{ij,y} & (u_j - u_i)\nabla W_{ij,z} \\ (v_j - v_i)\nabla W_{ij,x} & (v_j - v_i)\nabla W_{ij,y} & (v_j - v_i)\nabla W_{ij,z} \\ (w_j - w_i)\nabla W_{ij,x} & (w_j - w_i)\nabla W_{ij,y} & (w_j - w_i)\nabla W_{ij,z} \end{bmatrix} \tag{C.8}$$

$$(\nabla \mathbf{u}_i)^T = \sum_j V_j \nabla W_{ij} (\mathbf{u}_j - \mathbf{u}_i)^T \tag{C.9}$$

$$= \sum_j V_j \begin{bmatrix} \nabla W_{ij,x} \\ \nabla W_{ij,y} \\ \nabla W_{ij,z} \end{bmatrix} \begin{bmatrix} u_j - u_i & v_j - v_i & w_j - w_i \end{bmatrix} \tag{C.10}$$

$$= \sum_j V_j \begin{bmatrix} (u_j - u_i)\nabla W_{ij,x} & (v_j - v_i)\nabla W_{ij,x} & (w_j - w_i)\nabla W_{ij,x} \\ (u_j - u_i)\nabla W_{ij,y} & (v_j - v_i)\nabla W_{ij,y} & (w_j - w_i)\nabla W_{ij,y} \\ (u_j - u_i)\nabla W_{ij,z} & (v_j - v_i)\nabla W_{ij,z} & (w_j - w_i)\nabla W_{ij,z} \end{bmatrix} \tag{C.11}$$

$$\mathbf{u}_i + m\Delta t \sum_j \left(\frac{\mu_i}{\rho_i^2} \left(-\nabla \mathbf{u}_i - (\nabla \mathbf{u}_i)^T \right) + \frac{\mu_j}{\rho_j^2} \left(-\nabla \mathbf{u}_j - (\nabla \mathbf{u}_j)^T \right) \right) \nabla W_{ij} = \mathbf{u}_i^* \tag{C.12}$$

$$\mathbf{u}_i + \hat{m} \sum_j \left(\hat{\mu}_i \mathbf{Q}_{ij} + \hat{\mu}_j \mathbf{Q}_{jk} \right) \nabla W_{ij} = \mathbf{u}_i^* \tag{C.13}$$

$$\mathbf{Q}_{ij} = -\nabla \mathbf{u}_i - (\nabla \mathbf{u}_i)^T = \begin{bmatrix} q_{ij,xx} & q_{ij,xy} & q_{ij,xz} \\ q_{ij,xy} & q_{ij,yy} & q_{ij,yz} \\ q_{ij,xz} & q_{ij,yz} & q_{ij,zz} \end{bmatrix}, \tag{C.14}$$

$$q_{ij,xx} = 2 \sum_j V_j \nabla W_{ij,x} (u_i - u_j) = 2 \sum_j a_{ij,x} (u_i - u_j), \tag{C.15}$$

$$q_{ij,xy} = \sum_j V_j \left(\nabla W_{ij,y} (u_i - u_j) + \nabla W_{ij,x} (v_i - v_j) \right) = \sum_j \left(a_{ij,y} (u_i - u_j) + a_{ij,x} (v_i - v_j) \right) \tag{C.16}$$

$$q_{ij,xz} = \sum_j V_j \left(\nabla W_{ij,z} (u_i - u_j) + \nabla W_{ij,x} (w_i - w_j) \right) = \sum_j \left(a_{ij,z} (u_i - u_j) + a_{ij,x} (w_i - w_j) \right), \tag{C.17}$$

$$q_{ij,yy} = 2 \sum_j V_j \nabla W_{ij,y} (v_i - v_j) = 2 \sum_j a_{ij,y} (v_i - v_j), \tag{C.18}$$

$$q_{ij,yz} = \sum_j V_j \left(\nabla W_{ij,z} (y_i - y_j) + \nabla W_{ij,y} (w_i - w_j) \right) = \sum_j \left(a_{ij,z} (y_i - y_j) + a_{ij,y} (w_i - w_j) \right), \tag{C.19}$$

$$q_{ij,zz} = 2 \sum_j V_j \nabla W_{ij,z} (w_i - w_j) = 2 \sum_j a_{ij,z} (w_i - w_j), \tag{C.20}$$

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(2 \sum_j a_{ij,x}(u_i - u_j) \nabla W_{ij,x} + \\
\sum_j (a_{ij,y}(u_i - u_j) + a_{ij,x}(v_i - v_j)) \nabla W_{ij,y} \\
\sum_j (a_{ij,z}(u_i - u_j) + a_{ij,x}(w_i - w_j)) \nabla W_{ij,z} \Big) + \\
\hat{\mu}_j \Big(2 \sum_k a_{jk,x}(u_j - u_k) \nabla W_{ij,x} + \\
\sum_k (a_{jk,y}(u_j - u_k) + a_{jk,x}(v_j - v_k)) \nabla W_{ij,y} \\
\sum_k (a_{jk,z}(u_j - u_k) + a_{jk,x}(w_j - w_k)) \nabla W_{ij,z} \Big) \Bigg) \\
= u_i^\dagger. \qquad \text{(C.21)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(2 (\sum_j a_{ij,x} u_i - \sum_j a_{ij,x} u_j) \nabla W_{ij,x} + \\
(\sum_j a_{ij,y} u_i - \sum_j a_{ij,y} u_j + \sum_j a_{ij,x} v_i - \sum_j a_{ij,x} v_j) \nabla W_{ij,y} \\
(u_i \sum_j a_{ij,z} - \sum_j a_{ij,z} u_j + \sum_j a_{ij,x} w_i - \sum_j a_{ij,x} w_j) \nabla W_{ij,z} \Big) + \\
\hat{\mu}_j \Big(2 (\sum_k a_{jk,x} u_j - \sum_k a_{jk,x} u_k) \nabla W_{ij,x} + \\
(\sum_k a_{jk,y} u_j - \sum_k a_{jk,y} u_k + \sum_k a_{jk,x} v_j - \sum_k a_{jk,x} v_k) \nabla W_{ij,y} \\
(\sum_k a_{jk,z} u_j - \sum_k a_{jk,z} u_k + \sum_k a_{jk,x} w_j - \sum_k a_{jk,x} w_k) \nabla W_{ij,z} \Big) \Bigg) \\
= u_i^\dagger. \qquad \text{(C.22)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(& 2(u_i \sum_j a_{ij,x} - \sum_j a_{ij,x} u_j) \nabla W_{ij,x} + \\
& (u_i \sum_j a_{ij,y} - \sum_j a_{ij,y} u_j + v_i \sum_j a_{ij,x} - \sum_j a_{ij,x} v_j) \nabla W_{ij,y} \\
& (u_i \sum_j a_{ij,z} - \sum_j a_{ij,z} u_j + w_i \sum_j a_{ij,x} - \sum_j a_{ij,x} w_j) \nabla W_{ij,z} \Big) + \\
& \hat{\mu}_j \Big(2(u_j \sum_k a_{jk,x} - \sum_k a_{jk,x} u_k) \nabla W_{ij,x} + \\
& (u_j \sum_k a_{jk,y} - \sum_k a_{jk,y} u_k + v_j \sum_k a_{jk,x} - \sum_k a_{jk,x} v_k) \nabla W_{ij,y} \\
& (u_j \sum_k a_{jk,z} - \sum_k a_{jk,z} u_k + w_j \sum_k a_{jk,x} - \sum_k a_{jk,x} w_k) \nabla W_{ij,z} \Big) \Bigg) \\
& = u_i^\dagger. \qquad \text{(C.23)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(& 2(u_i \alpha_{ij,x} - \sum_j a_{ij,x} u_j) \nabla W_{ij,x} + \\
& (u_i \alpha_{ij,y} - \sum_j a_{ij,y} u_j + v_i \alpha_{ij,x} - \sum_j a_{ij,x} v_j) \nabla W_{ij,y} \\
& (u_i \alpha_{ij,z} - \sum_j a_{ij,z} u_j + w_i \alpha_{ij,x} - \sum_j a_{ij,x} w_j) \nabla W_{ij,z} \Big) + \\
& \hat{\mu}_j \Big(2(u_j \alpha_{jk,x} - \sum_k a_{jk,x} u_k) \nabla W_{ij,x} + \\
& (u_j \alpha_{jk,y} - \sum_k a_{jk,y} u_k + v_j \alpha_{jk,x} - \sum_k a_{jk,x} v_k) \nabla W_{ij,y} \\
& (u_j \alpha_{jk,z} - \sum_k a_{jk,z} u_k + w_j \alpha_{jk,x} - \sum_k a_{jk,x} w_k) \nabla W_{ij,z} \Big) \Bigg) \\
& = u_i^\dagger. \qquad \text{(C.24)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + \hat{m} \sum_j \Bigg(\hat{\mu}_i \Big(& 2\nabla W_{ij,x} u_i \alpha_{ij,x} - 2\nabla W_{ij,x} \sum_j a_{ij,x} u_j + \\
& \nabla W_{ij,y} u_i \alpha_{ij,y} - \nabla W_{ij,y} \sum_j a_{ij,y} u_j + \nabla W_{ij,y} v_i \alpha_{ij,x} - \nabla W_{ij,y} \sum_j a_{ij,x} v_j \\
& \nabla W_{ij,z} u_i \alpha_{ij,z} - \nabla W_{ij,z} \sum_j a_{ij,z} u_j + \nabla W_{ij,z} w_i \alpha_{ij,x} - \nabla W_{ij,z} \sum_j a_{ij,x} w_j \Big) + \\
& \hat{\mu}_j \Big(2\nabla W_{ij,x} u_j \alpha_{jk,x} - 2\nabla W_{ij,x} \sum_k a_{jk,x} u_k) + \\
& \nabla W_{ij,y} u_j \alpha_{jk,y} - \nabla W_{ij,y} \sum_k a_{jk,y} u_k + \nabla W_{ij,y} v_j \alpha_{jk,x} - \nabla W_{ij,y} \sum_k a_{jk,x} v_k \\
& \nabla W_{ij,z} u_j \alpha_{jk,z} - \nabla W_{ij,z} \sum_k a_{jk,z} u_k + \nabla W_{ij,z} w_j \alpha_{jk,x} - \nabla W_{ij,z} \sum_k a_{jk,x} w_k \Big) \Bigg) \\
& = u_i^\dagger. \qquad \text{(C.25)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + \hat{m}\hat{\mu}_i \sum_j \Big(& 2\nabla W_{ij,x} u_i \alpha_{ij,x} - 2\nabla W_{ij,x} \sum_j a_{ij,x} u_j + \\
& \nabla W_{ij,y} u_i \alpha_{ij,y} - \nabla W_{ij,y} \sum_j a_{ij,y} u_j + \nabla W_{ij,y} v_i \alpha_{ij,x} - \nabla W_{ij,y} \sum_j a_{ij,x} v_j \\
& \nabla W_{ij,z} u_i \alpha_{ij,z} - \nabla W_{ij,z} \sum_j a_{ij,z} u_j + \nabla W_{ij,z} w_i \alpha_{ij,x} - \nabla W_{ij,z} \sum_j a_{ij,x} w_j \Big) + \\
& \hat{m} \sum_j \hat{\mu}_j \Big(2\nabla W_{ij,x} u_j \alpha_{jk,x} - 2\nabla W_{ij,x} \sum_k a_{jk,x} u_k + \\
& \nabla W_{ij,y} u_j \alpha_{jk,y} - \nabla W_{ij,y} \sum_k a_{jk,y} u_k + \nabla W_{ij,y} v_j \alpha_{jk,x} - \nabla W_{ij,y} \sum_k a_{jk,x} v_k \\
& \nabla W_{ij,z} u_j \alpha_{jk,z} - \nabla W_{ij,z} \sum_k a_{jk,z} u_k + \nabla W_{ij,z} w_j \alpha_{jk,x} - \nabla W_{ij,z} \sum_k a_{jk,x} w_k \Big) \\
& = u_i^\dagger. \qquad \text{(C.26)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + 2\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,x} u_i \alpha_{ij,x} - 2\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,x} \sum_j a_{ij,x} u_j + \\
\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,y} u_i \alpha_{ij,y} - \hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,y} \sum_j a_{ij,y} u_j + \\
\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,y} v_i \alpha_{ij,x} - \hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,y} \sum_j a_{ij,x} v_j + \\
\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,z} u_i \alpha_{ij,z} - \hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,z} \sum_j a_{ij,z} u_j + \\
\hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,z} w_i \alpha_{ij,x} - \hat{m}\hat{\mu}_i \sum_j \nabla W_{ij,z} \sum_j a_{ij,x} w_j + \\
2\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,x} u_j \alpha_{jk,x} - 2\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,x} \sum_k a_{jk,x} u_k + \\
\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,y} u_j \alpha_{jk,y} - \hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,y} \sum_k a_{jk,y} u_k + \\
\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,y} v_j \alpha_{jk,x} - \hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,y} \sum_k a_{jk,x} v_k + \\
\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,z} u_j \alpha_{jk,z} - \hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,z} \sum_k a_{jk,z} u_k + \\
\hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,z} w_j \alpha_{jk,x} - \hat{m} \sum_j \hat{\mu}_j \nabla W_{ij,z} \sum_k a_{jk,x} w_k \\
= u_i^{\dagger}. \qquad \text{(C.27)}
\end{aligned}
$$

$$
\begin{aligned}
u_i + 2\hat{m}\hat{\mu}_i\omega_{ij,x}u_i\alpha_{ij,x} - 2\hat{m}\hat{\mu}_i\omega_{ij,x}\sum_j a_{ij,x}u_j + \\
\hat{m}\hat{\mu}_i\omega_{ij,y}u_i\alpha_{ij,y} - \hat{m}\hat{\mu}_i\omega_{ij,y}\sum_j a_{ij,y}u_j + \\
\hat{m}\hat{\mu}_i\omega_{ij,y}v_i\alpha_{ij,x} - \hat{m}\hat{\mu}_i\omega_{ij,y}\sum_j a_{ij,x}v_j + \\
\hat{m}\hat{\mu}_i\omega_{ij,z}u_i\alpha_{ij,z} - \hat{m}\hat{\mu}_i\omega_{ij,z}\sum_j a_{ij,z}u_j + \\
\hat{m}\hat{\mu}_i\omega_{ij,z}w_i\alpha_{ij,x} - \hat{m}\hat{\mu}_i\omega_{ij,z}\sum_j a_{ij,x}w_j + \\
2\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,x}u_j\alpha_{jk,x} - 2\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,x}\sum_k a_{jk,x}u_k + \\
\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}u_j\alpha_{jk,y} - \hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}\sum_k a_{jk,y}u_k + \\
\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}v_j\alpha_{jk,x} - \hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}\sum_k a_{jk,x}v_k + \\
\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}u_j\alpha_{jk,z} - \hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}\sum_k a_{jk,z}u_k + \\
\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}w_j\alpha_{jk,x} - \hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}\sum_k a_{jk,x}w_k \\
= u_i^{\dagger}. \qquad \text{(C.28)}
\end{aligned}
$$

$$
\begin{aligned}
&(1+\hat{m}\hat{\mu}_i(2\omega_{ij,x}\alpha_{ij,x}+\omega_{ij,y}\alpha_{ij,y}+\omega_{ij,z}\alpha_{ij,z}))u_i + \\
&\hat{m}\hat{\mu}_i\omega_{ij,y}\alpha_{ij,x}v_i + \\
&\hat{m}\hat{\mu}_i\omega_{ij,z}\alpha_{ij,x}w_i + \\
&\hat{m}\hat{\mu}_i(-2\omega_{ij,x}\sum_j a_{ij,x}u_j-\omega_{ij,y}\sum_j a_{ij,y}u_j-\omega_{ij,z}\sum_j a_{ij,z}u_j) + \\
&\hat{m}\hat{\mu}_i(-\omega_{ij,y}\sum_j a_{ij,x}v_j) + \\
&\hat{m}\hat{\mu}_i(-\omega_{ij,z}\sum_j a_{ij,x}w_j) \qquad \text{(C.29)} \\
&\sum_j \hat{\mu}_j(2\nabla W_{ij,x}\alpha_{jk,x}+\nabla W_{ij,y}\alpha_{jk,y}\nabla W_{ij,z}\alpha_{jk,z})u_j \\
&\sum_j \hat{\mu}_j\nabla W_{ij,y}\alpha_{jk,x}v_j \\
&\sum_j \hat{\mu}_j\nabla W_{ij,z}\alpha_{jk,x}w_j \\
&\sum_j \hat{\mu}_j(-2\nabla W_{ij,x}\sum_k a_{jk,x}u_k-\nabla W_{ij,y}\sum_k a_{jk,y}u_k-\nabla W_{ij,z}\sum_k a_{jk,z}u_k) + \\
&-\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}\sum_k a_{jk,x}v_k + \\
&-\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}\sum_k a_{jk,x}w_k \\
&= u_i^{\dagger}. \qquad \text{(C.30)}
\end{aligned}
$$

Then, I further convert Eq. (4.9) into the following equation with coefficients $c_{u_iu_i}, c_{v_iu_i}, c_{w_iu_i}, c_{u_ju_i}, c_{v_ju_i}, c_{w_ju_i}, c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$:

$$
\begin{bmatrix} c_{u_iu_i} \\ c_{v_iu_i} \\ c_{w_iu_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_ju_i} \\ c_{v_ju_i} \\ c_{w_ju_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_ku_i} \\ c_{v_ku_i} \\ c_{w_ku_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = u_i^{\dagger}, \qquad \text{(C.31)}
$$

$$c_{u_iu_i} = 1 + \hat{m}\hat{\mu}_i \left(2\omega_{ij,x}\alpha_{ij,x} + \omega_{ij,y}\alpha_{ij,y} + \omega_{ij,z}\alpha_{ij,z}\right),$$

$$c_{v_iu_i} = \hat{m}\hat{\mu}_i\omega_{ij,y}\alpha_{ij,x}, \tag{C.32}$$

$$c_{w_iu_i} = \hat{m}\hat{\mu}_i\omega_{ij,z}\alpha_{ij,x}, \tag{C.33}$$

$$c_{u_ju_i} = \hat{m}\big(-\hat{\mu}_i(2a_{ij,x}\omega_{ij,x} + a_{ij,y}\omega_{ij,y} + a_{ij,z}\omega_{ij,z}) +$$
$$\hat{\mu}_j(2\nabla W_{ij,x}\alpha_{jk,x} + \nabla W_{ij,y}\alpha_{jk,y} + \nabla W_{ij,z}\alpha_{jk,z})\big), \tag{C.34}$$

$$c_{v_ju_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,y} + \hat{\mu}_j\nabla W_{ij,y}\alpha_{jk,x}\right), \tag{C.35}$$

$$c_{w_ju_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,z} + \hat{\mu}_j\nabla W_{ij,z}\alpha_{jk,x}\right),$$

$$c_{u_ku_i} = -\hat{m}\sum_j \hat{\mu}_j(2\nabla W_{ij,x}a_{jk,x} + \nabla W_{ij,y}a_{jk,y} + \nabla W_{ij,z}a_{jk,z}), \tag{C.36}$$

$$c_{v_ku_i} = -\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,y}a_{jk,x}, \tag{C.37}$$

$$c_{w_ku_i} = -\hat{m}\sum_j \hat{\mu}_j\nabla W_{ij,z}a_{jk,x}, \tag{C.38}$$

where $\alpha_{ij} = [\alpha_{ij,x}, \alpha_{ij,y}, \alpha_{ij,z}]^T = \sum_j \mathbf{a}_{ij}$ and $\omega_{ij} = [\omega_{ij,x}, \omega_{ij,y}, \omega_{ij,z}]^T = \sum_j \nabla W_{ij}$. I use $c_{u_iu_i}$ to denote a coefficient of u_i to u_i, and $c_{v_iu_i}$ a coefficient of v_i to u_i, and similarly define other coefficients.

Similarly, for v_i, I compute coefficients $c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_kv_i}, c_{v_kv_i}$, and $c_{w_kv_i}$:

$$\begin{bmatrix} c_{u_iv_i} \\ c_{v_iv_i} \\ c_{w_iv_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_jv_i} \\ c_{v_jv_i} \\ c_{w_jv_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_kv_i} \\ c_{v_kv_i} \\ c_{w_kv_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = v_i^\dagger, \tag{C.39}$$

$$c_{u_i,v_i} = \hat{m}\hat{\mu}_i\omega_{ij,x}\alpha_{ij,y}, \tag{C.40}$$

$$c_{v_i,v_i} = 1 + \hat{m}\hat{\mu}_i\left(\omega_{ij,x}\alpha_{ij,x} + 2\omega_{ij,y}\alpha_{ij,y} + \omega_{ij,z}\alpha_{ij,z}\right), \tag{C.41}$$

$$c_{z_i,v_i} = \hat{m}\hat{\mu}_i\omega_{ij,z}\alpha_{ij,y}, \tag{C.42}$$

$$c_{u_j v_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,y}\omega_{ij,x} + \hat{\mu}_j \nabla W_{ij,x}\alpha_{jk,y}\right), \tag{C.43}$$

$$c_{v_j v_i} = \hat{m}\Bigg(-\hat{\mu}_i(a_{ij,x}\omega_{ij,x} + 2a_{ij,y}\omega_{ij,y} + a_{ij,z}\omega_{ij,z}) + \hat{\mu}_j(\nabla W_{ij,x}\alpha_{jk,x} + 2\nabla W_{ij,y}\alpha_{jk,y} + \nabla W_{ij,z}\alpha_{jk,z})\Bigg), \tag{C.44}$$

$$c_{w_j v_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,y}\omega_{ij,z} + \hat{\mu}_j \nabla W_{ij,z}\alpha_{jk,y}\right), \tag{C.45}$$

$$c_{u_k v_i} = -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,x} a_{jk,y}, \tag{C.46}$$

$$c_{v_k v_i} = -\hat{m}\sum_j \hat{\mu}_j (\nabla W_{ij,x} a_{jk,x} + 2\nabla W_{ij,y} a_{jk,y} + \nabla W_{ij,z} a_{jk,z}), \tag{C.47}$$

$$c_{w_k v_i} = -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,z} a_{jk,y}. \tag{C.48}$$

Similarly, for w_i, I compute coefficients $c_{u_i w_i}, c_{v_i w_i}, c_{w_i w_i}, c_{u_j w_i}, c_{v_j w_i}, c_{w_j w_i}, c_{u_k w_i}, c_{v_k w_i}$, and $c_{w_k w_i}$:

$$\begin{bmatrix} c_{u_i w_i} \\ c_{v_i w_i} \\ c_{w_i w_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_j w_i} \\ c_{v_j w_i} \\ c_{w_j w_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_k w_i} \\ c_{v_k w_i} \\ c_{w_k w_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = w_i^\dagger, \tag{C.49}$$

$$c_{u_i w_i} = \hat{m}\hat{\mu}_i \omega_{ij,x}\alpha_{ij,z}, \tag{C.50}$$

$$c_{v_i w_i} = \hat{m}\hat{\mu}_i \omega_{ij,y}\alpha_{ij,z}, \tag{C.51}$$

$$c_{w_i w_i} = 1 + \hat{m}\hat{\mu}_i\left(\omega_{ij,x}\alpha_{ij,x} + \omega_{ij,y}\alpha_{ij,y} + 2\omega_{ij,z}\alpha_{ij,z}\right), \tag{C.52}$$

$$c_{u_j w_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,z}\omega_{ij,x} + \hat{\mu}_j \nabla W_{ij,x}\alpha_{jk,z}\right), \tag{C.53}$$

$$c_{v_j w_i} = \hat{m}\left(-\hat{\mu}_i a_{ij,z}\omega_{ij,y} + \hat{\mu}_j \nabla W_{ij,y}\alpha_{jk,z}\right), \tag{C.54}$$

$$c_{w_j w_i} = \hat{m}\Big(-\hat{\mu}_i(a_{ij,x}\omega_{ij,x} + a_{ij,y}\omega_{ij,y} + 2a_{ij,z}\omega_{ij,z}) + \hat{\mu}_j(\nabla W_{ij,x}\alpha_{jk,x} + \nabla W_{ij,y}\alpha_{jk,y} + 2\nabla W_{ij,z}\alpha_{jk,z})\Big), \tag{C.55}$$

$$c_{u_k w_i} = -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,x} a_{jk,z}, \tag{C.56}$$

$$c_{v_k w_i} = -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,y} a_{jk,z}, \tag{C.57}$$

$$c_{w_k w_i} = -\hat{m}\sum_j \hat{\mu}_j(\nabla W_{ij,x} a_{jk,x} + \nabla W_{ij,y} a_{jk,y} + 2\nabla W_{ij,z} a_{jk,z}). \tag{C.58}$$

APPENDIX D: DETAILS FOR THE VISCOUS FLUID SOLVER

D.1 Statistics

Table D.1 summarizes the simulation condition and performance for all the results. I note that the computational overhead for my geometrically consistent volume estimation is relatively small compared to the inconsistent volume estimation and the supersampling. Similarly, my strong two-way coupling requires almost the same amount of time for one-way and weak two-way coupling methods. My position-correction method can better enforce the uniform distributions of particles leading to the smaller number of neighbor particles, and consequently, my method is at least 6 times faster than the distance-based position correction and is 2.5 times faster than the density-based position correction.

D.2 Derivation of Terminal Velocity for Solid Spheres

The analytical solution for the terminal velocity of a spherical solid in viscous fluids can be derived by equating the drag, buoyant, and gravity forces. When the Reynolds number $Re = 2\frac{\rho_f ||\mathbf{V}||_2 r}{\eta}$ (ρ_f: fluid density, $\mathbf{V}$: solid velocity, r: solid radius, η: dynamic viscosity of fluids) is sufficiently low (typically $Re \ll 1$) and the domain is open boundaries (i.e., domain boundaries are far apart, and their influence to the solid is negligible), the drag force due to the viscosity $\mathbf{F}_d$ is $\mathbf{F}_d = 6\pi\eta r\mathbf{V}$ according to the Stokes' law. Since the sum of buoyant and gravity forces is $\mathbf{F}_g = \frac{4}{3}(\rho_s - \rho_f)\mathbf{g}\pi r^3$ (ρ_s: solid density, $\mathbf{g}$: gravity), the analytical terminal velocity $\hat{\mathbf{V}}^\infty$ is given by

$$\hat{\mathbf{V}}^\infty = \frac{2}{9}\frac{\rho_s - \rho_f}{\eta}\mathbf{g}r^2.$$

I note that this equation is valid only with a sufficiently low Reynolds number and open boundaries.

D.3 Two-Way Fluid-Solid Coupling

To validate the accuracy of the two-way fluid-solid coupling method compared to one-way coupling and weak two-way coupling in a wide range of viscosity values, I performed several experiments using a scenario, where a solid ball is falling inside of viscous fluids. In this experiment, I use $\eta = 1.0 \times 10^1, 1.0 \times 10^2, 1.0 \times$

$10^3, 1.0 \times 10^4$, and 1.0×10^5 $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$, and visual results and solid velocity profiles are shown in Figures D.1, D.2, D.3, D.4, and D.5, respectively.

Except for the case with viscosity $\eta = 1.0 \times 10^1$ $\mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$ (Figure D.1), where the Stokes' law and thus the analytical terminal velocity are not valid due to the high Reynolds number, the velocities of the solid balls simulated with strong two-way coupling are in good agreement with the analytical solutions. I note that while it requires a small amount of time for the simulated solid balls to reach the equilibrium, the solid ball for the analytical solution is directly moved from the beginning with the velocity given by the Stokes' law. As such, the height of the balls can be different for the simulation and analytical solution (e.g., in Figure D.2).

By contrast, the velocities of solid balls simulated with one-way coupling and weak two-way coupling significantly deviate from the analytical solutions, unnaturally oscillate, and do not sufficiently reflect the differences of viscosity values. In addition, the incorrect behaviors of the solid ball with one-way and weak two-way coupling unnaturally deform the viscous fluid blocks (see the top of the blocks in the figures).

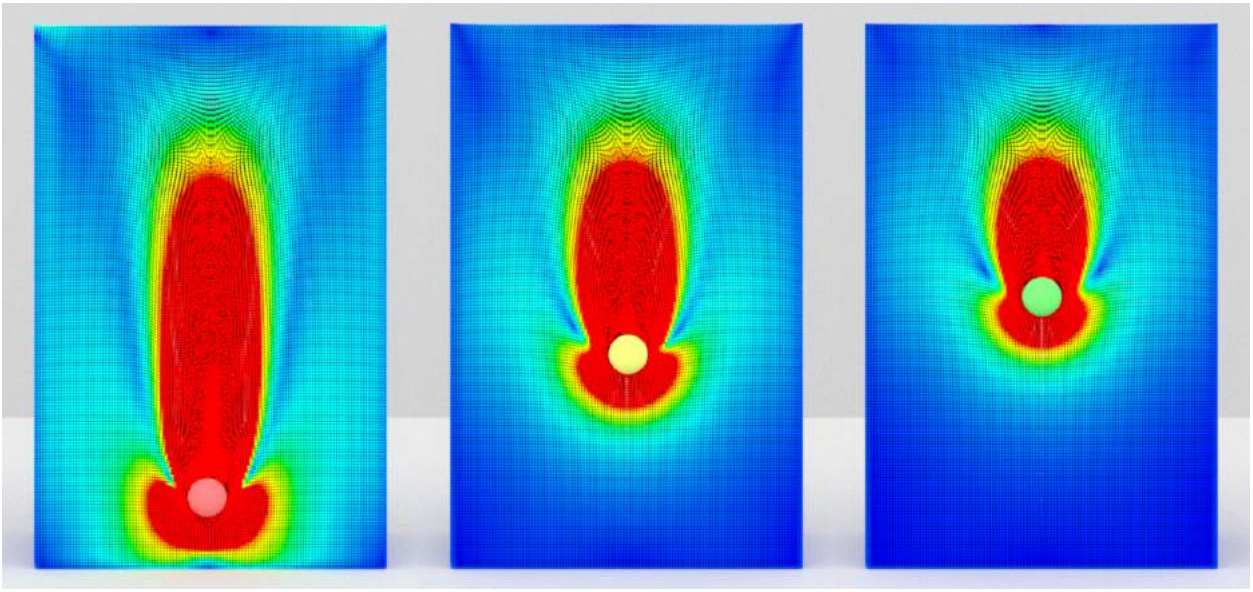
D.4 Position Correction

Figure D.6 compares my position-correction method with methods using no position corrections and distance-based position corrections, and I use up to 50 iterations for my method and the distance-based position correction method. In this scene, a bulk of highly viscous fluids is successively compressed by circular plates with multiple holes.

The method with no position corrections can easily lose fluid volumes, and the viscous fluid does not reach the top. While the method with the distance-based position correction can preserve the volume better reaching the top, my method enables more volumes to reach the top. The color-coding for simulation particles also clarify that my method can resolve the compression of particles better compared to the distance-based position corrections. Additionally, I note that compared to my method, the distance-based position correction method can be more costly due to the large number of neighboring particles.

Table D.1: Simulation conditions and performance results. "Volume" represents which method is used for volume computation. "Coupling" represents a scheme used for the fluid-solid coupling. "Uniform" represents a scheme used to enforce the uniform distributions of particles, and the number of maximum iterations. $t_{\mathrm{vol}}, t_{\mathrm{pres}}, t_{\mathrm{visc}}, t_{\mathrm{dens}}, t_{\mathrm{rest}}$, and t_{total} represent computation time in seconds per frame for volume fraction computation, pressure solve, viscosity solve, density solve, rest (e.g., data transfers, particle advection, velocity extrapolation), and total, respectively. * indicates figure numbers in chapter 5. The computational time for the inconsistent volume estimation, supersampling, and consistent volume estimation are comparable (in orange). My strong two-way coupling requires about the same amount of time for one-way and weak two-way coupling (in cyan). My position-correction method is at least 6 times faster than distance-based and 2.5 times faster than density-based position corrections (in magenta).

Scene	Grid resolution	Particles	η kg/(s · m)	Volume	Coupling	Uniform	t_{vol}	t_{pres}	t_{visc}	t_{dens}	t_{rest}	t_{total}
Fig. 1* (leftmost)	$128 \times 128 \times 128$	2,798.2k	1.0×10^3	Mine	One-way	Mine/3	11.2	2.2	**26.0**	16.2	10.1	65.7
Fig. 1* (left)	$128 \times 128 \times 128$	2,798.2k	1.0×10^3	Mine	Weak	Mine/3	10.1	2.0	**20.4**	15.3	8.6	56.4
Fig. 1* (right)	$128 \times 128 \times 128$	2,798.2k	1.0×10^3	Mine	Strong	Mine/3	8.9	1.8	**20.4**	9.0	5.7	45.7
Fig. 4* (left)	$192 \times 192 \times 192$	800.5k	1.0×10^1	Inconsistent	N/A	Mine/3	**11.0**	4.7	49.5	5.5	19.4	90.2
Fig. 4* (middle)	$192 \times 192 \times 192$	800.5k	1.0×10^1	Supersampling	N/A	Mine/3	**11.0**	5.7	47.1	5.3	19.9	89.0
Fig. 4* (right)	$192 \times 192 \times 192$	800.5k	1.0×10^1	Mine	N/A	Mine/3	**12.5**	5.2	51.7	5.4	19.6	94.5
Fig. 5* (left)	$144 \times 96 \times 96$	3,721.8k	up to 1.0×10^8	Mine	One-way	Mine/3	12.3	3.3	**32.7**	23.8	32.6	104.7
Fig. 5* (middle)	$144 \times 96 \times 96$	3,721.8k	up to 1.0×10^8	Mine	Weak	Mine/3	15.3	4.1	**44.0**	33.3	40.3	137.0
Fig. 5* (right)	$144 \times 96 \times 96$	3,721.8k	up to 1.0×10^8	Mine	Strong	Mine/3	11.2	3.0	**42.2**	21.3	26.7	104.4
Fig. 6* (middle)	$128 \times 128 \times 128$	up to 344.2k	1.0×10^3	Mine	N/A	Dist/50	12.9	0.6	5.9	**75.6**	3.8	98.8
Fig. 6* (middle)	$128 \times 128 \times 128$	up to 344.2k	1.0×10^3	Mine	N/A	Dens/50	15.3	0.7	7.0	**29.7**	4.7	57.4
Fig. 6* (right)	$128 \times 128 \times 128$	up to 344.2k	1.0×10^3	Mine	N/A	Mine/50	11.7	0.6	5.6	**11.7**	3.5	33.1
Fig. 7* (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	Mine	One-way	Mine/3	3.3	1.3	**10.6**	16.9	9.2	40.7
Fig. 7* (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	Mine	Weak	Mine/3	3.0	1.2	**9.6**	16.2	3.5	33.5
Fig. 7* (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	Mine	Strong	Mine/3	2.7	1.1	**9.7**	7.3	2.9	23.7
Fig. 8* (left)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	None/0	43.4	6.6	38.6	**0.0**	71.5	160.1
Fig. 8* (middle)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	Dist/3	42.3	9.3	46.5	**39.3**	74.2	211.6
Fig. 8* (right)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	mine/3	39.8	8.6	43.6	**22.1**	72.9	187.1
Fig. 9* (left)	$192 \times 384 \times 192$	up to 1,805.0k	1.0×10^1	mine	Strong	mine/3	152.1	13.5	81.8	11.5	82.8	341.8
Fig. 9* (middle)	$192 \times 384 \times 192$	up to 1,805.0k	1.0×10^2	mine	Strong	mine/3	129.6	11.0	95.4	9.4	69.9	315.2
Fig. 9* (right)	$192 \times 384 \times 192$	up to 1,805.0k	1.0×10^3	mine	Strong	mine/3	84.8	6.7	101.1	9.6	45.1	247.1
Fig. 10*	$128 \times 128 \times 128$	2,756.6k	1.0×10^2	mine	Strong	mine/3	30.0	5.5	70.8	28.4	43.5	178.2
Fig. 11* (left)	$128 \times 128 \times 128$	258.5k	3.0×10^2	mine	Strong	mine/50	17.1	1.5	11.9	16.1	10.8	57.4
Fig. D.1 (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^1	mine	One-way	mine/3	4.4	1.8	**9.7**	21.0	12.5	49.5
Fig. D.1 (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^1	mine	Weak	mine/3	3.0	1.2	**6.4**	16.3	5.7	32.6
Fig. D.1 (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^1	mine	Strong	mine/3	2.8	1.1	**5.9**	15.1	4.1	29.0
Fig. D.2 (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^2	mine	One-way	mine/3	4.3	1.7	**9.3**	22.3	10.2	47.8
Fig. D.2 (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^2	mine	Weak	mine/3	3.0	1.2	**7.6**	16.0	3.5	31.3
Fig. D.2 (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^2	mine	Strong	mine/3	2.8	1.2	**7.7**	8.1	2.9	22.8
Fig. D.3 (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	mine	One-way	mine/3	3.3	1.3	**10.6**	16.9	9.2	40.7
Fig. D.3 (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	mine	Weak	mine/3	3.0	1.2	**9.6**	16.2	3.5	33.5
Fig. D.3 (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^3	mine	Strong	mine/3	2.7	1.1	**9.7**	7.3	2.9	23.7
Fig. D.4 (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^4	mine	One-way	mine/3	3.7	1.4	**13.2**	17.6	10.5	46.3
Fig. D.4 (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^4	mine	Weak	mine/3	3.1	1.1	**11.3**	16.8	3.7	36.0
Fig. D.4 (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^4	mine	Strong	mine/3	2.6	1.0	**11.8**	6.9	2.7	24.9
Fig. D.5 (left)	$64 \times 128 \times 64$	2,405.0k	1.0×10^5	mine	One-way	mine/3	3.6	1.6	**20.0**	18.0	9.3	52.5
Fig. D.5 (middle)	$64 \times 128 \times 64$	2,405.0k	1.0×10^5	mine	Weak	mine/3	3.2	1.2	**13.1**	17.6	3.8	39.0
Fig. D.5 (right)	$64 \times 128 \times 64$	2,405.0k	1.0×10^5	mine	Strong	mine/3	2.8	1.0	**11.8**	7.4	2.9	25.8
Fig. D.6 (left)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	None/0	43.4	6.6	38.6	**0.0**	71.5	160.1
Fig. D.6 (middle)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	Dist/50	45.7	10.5	54.7	**538.6**	87.1	735.5
Fig. D.6 (right)	$128 \times 128 \times 128$	1,891.5k	1.0×10^8	mine	N/A	mine/50	39.7	10.2	48.3	**190.9**	78.8	367.8



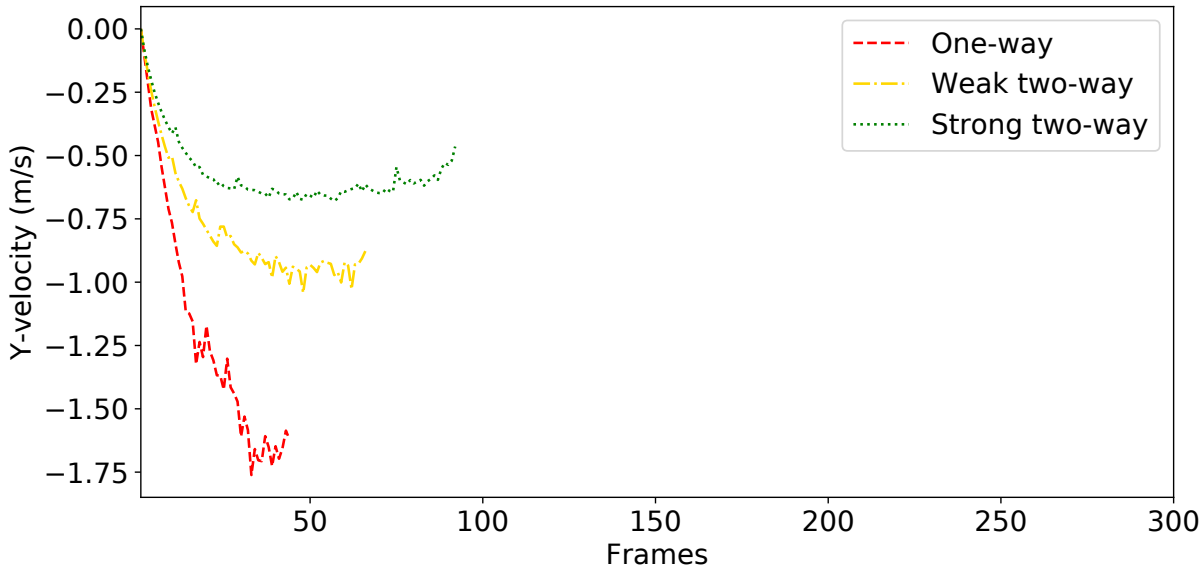
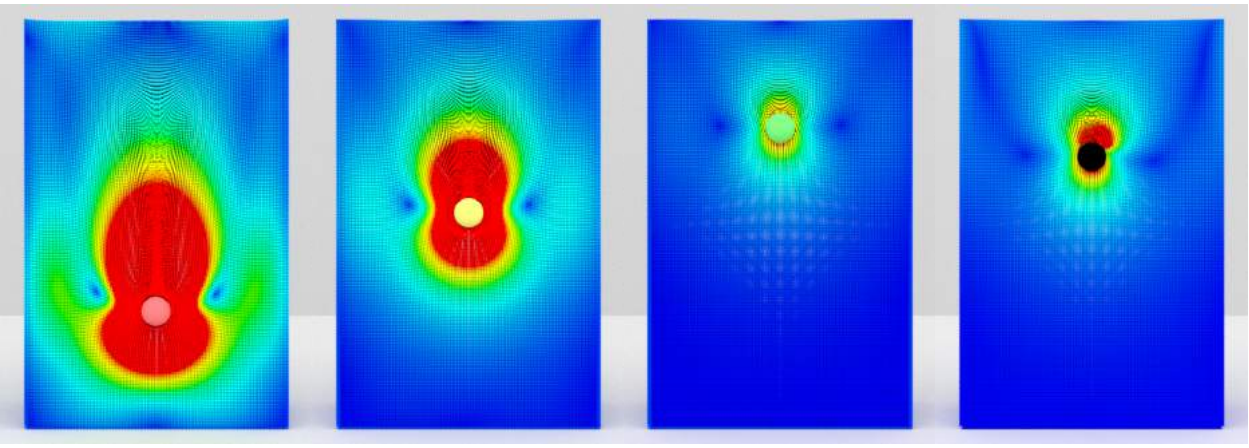


Figure D.1: A solid ball falling inside of fluids with viscosity $\eta = 1.0 \times 10^1$ kg/(s · m). (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. Red, green, and blue particles represent large, medium, and small velocity magnitudes, respectively. (Bottom) Profile of the y-directional velocity of the solid balls. Note that due to the high Reynolds number, the analytical solution is not valid and thus the figure and plot for the analytical solution is excluded.



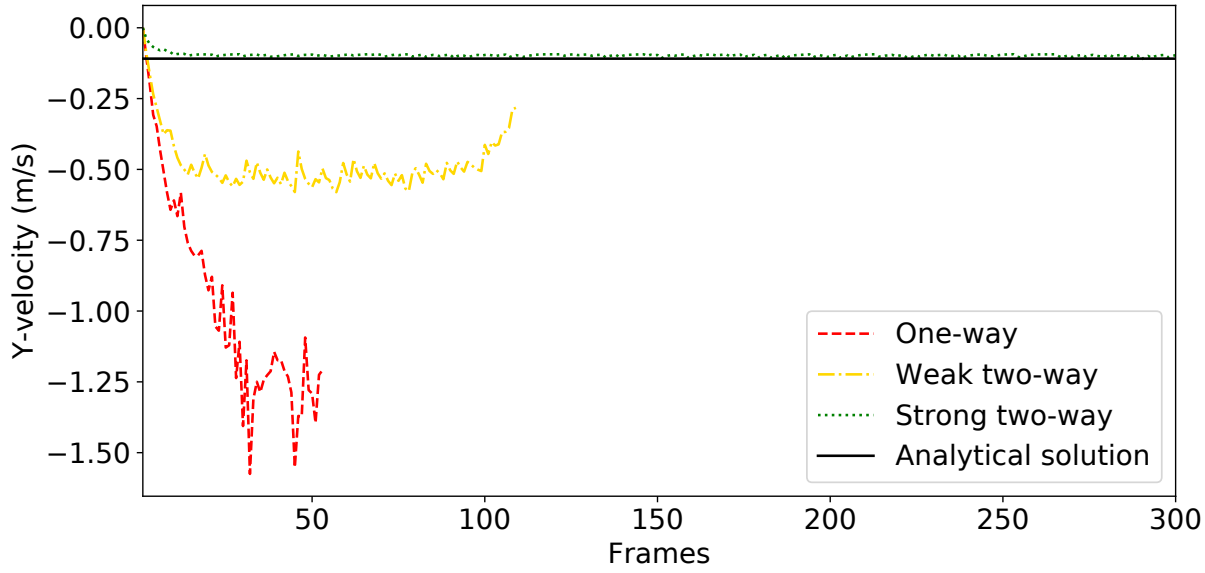


Figure D.2: A solid ball falling inside of fluids with viscosity value $\eta = 1.0 \times 10^2\ \mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. (Bottom) Profile of the y-directional velocity of the solid balls. My strong two-way coupling gives solid velocities very close to the analytical solution while the solid velocities given with one-way and weak two-way coupling significantly deviate from the analytical solution.

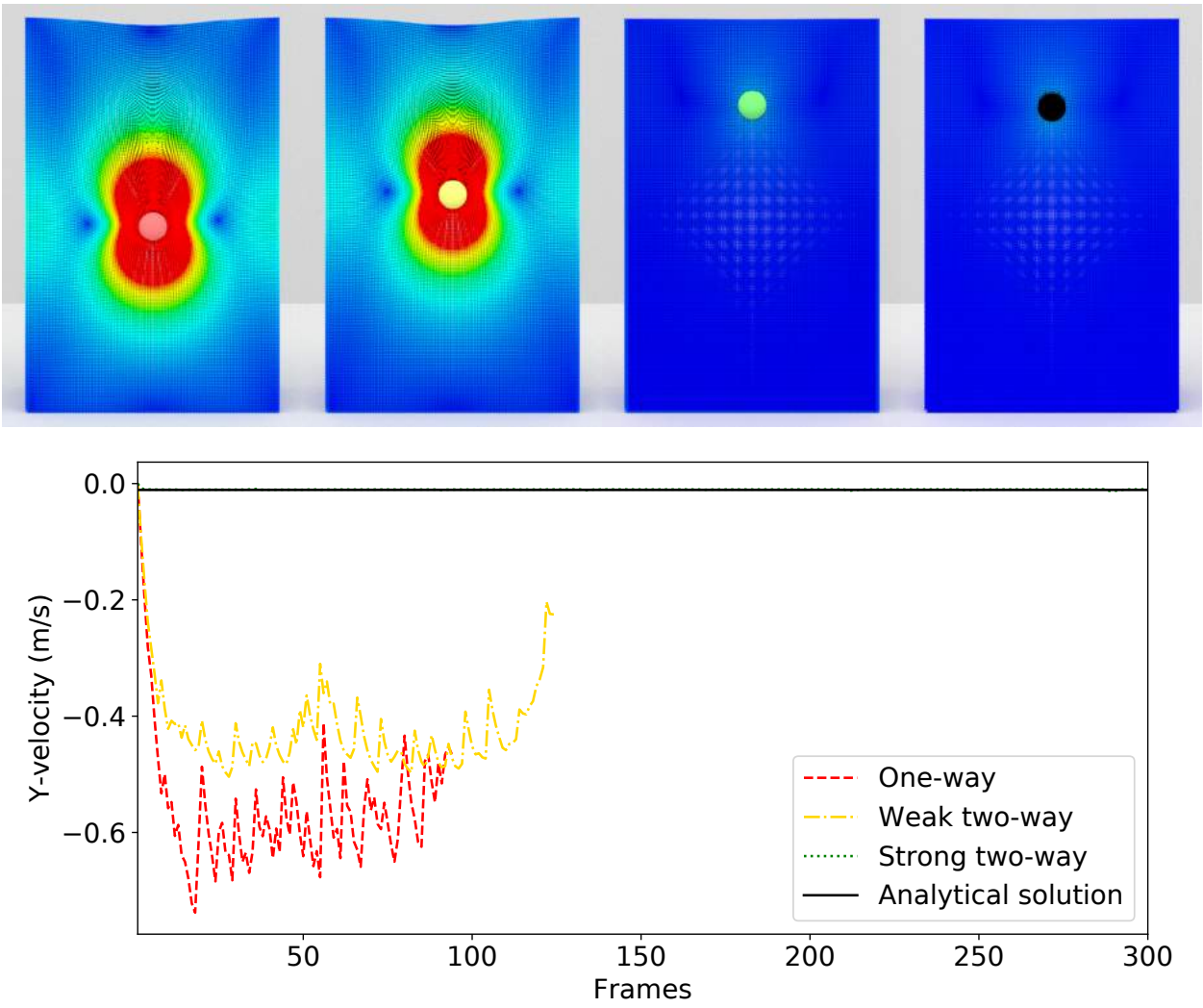
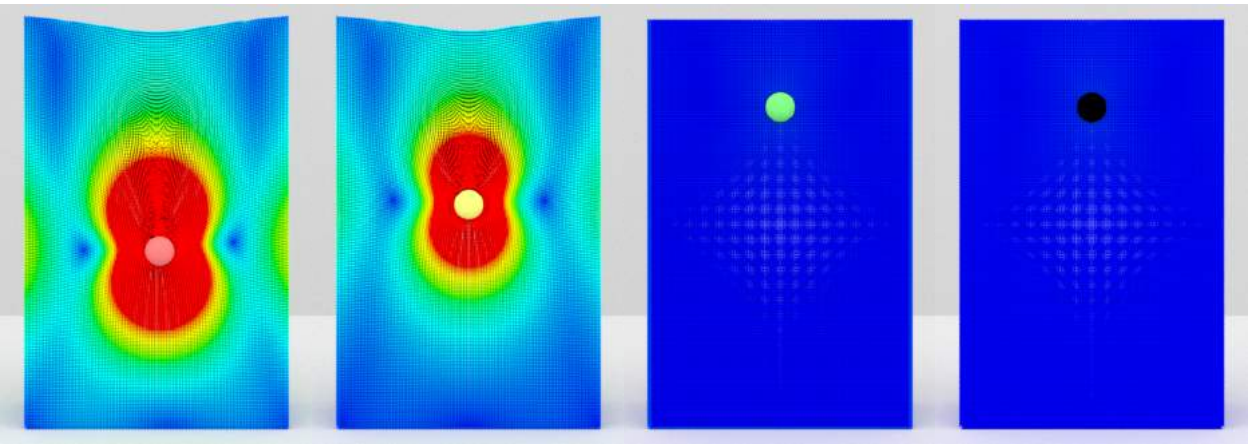


Figure D.3: A solid ball falling inside of fluids with viscosity values $\eta = 1.0 \times 10^3\ \mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. (Bottom) Profile of the y-directional velocity of the solid balls. My strong two-way coupling gives solid velocities very close to the analytical solution while the solid velocities given with one-way and weak two-way coupling significantly deviate from the analytical solution.



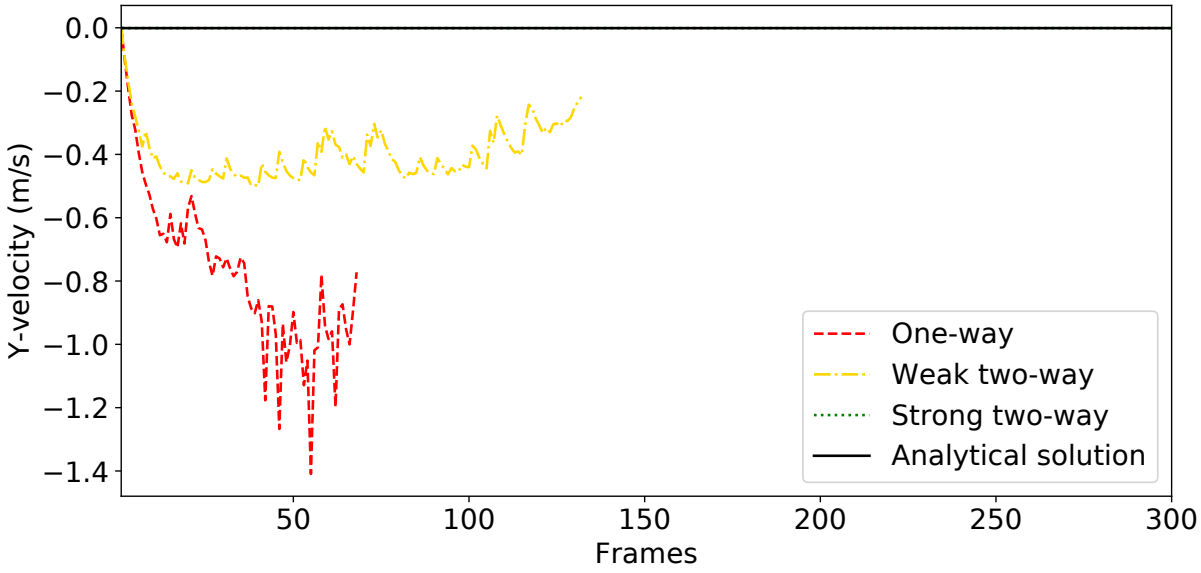


Figure D.4: A solid ball falling inside of fluids with viscosity values $\eta = 1.0 \times 10^4\ \mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. (Bottom) Profile of the y-directional velocity of the solid balls. My strong two-way coupling gives solid velocities very close to the analytical solution while the solid velocities given with one-way and weak two-way coupling significantly deviate from the analytical solution.

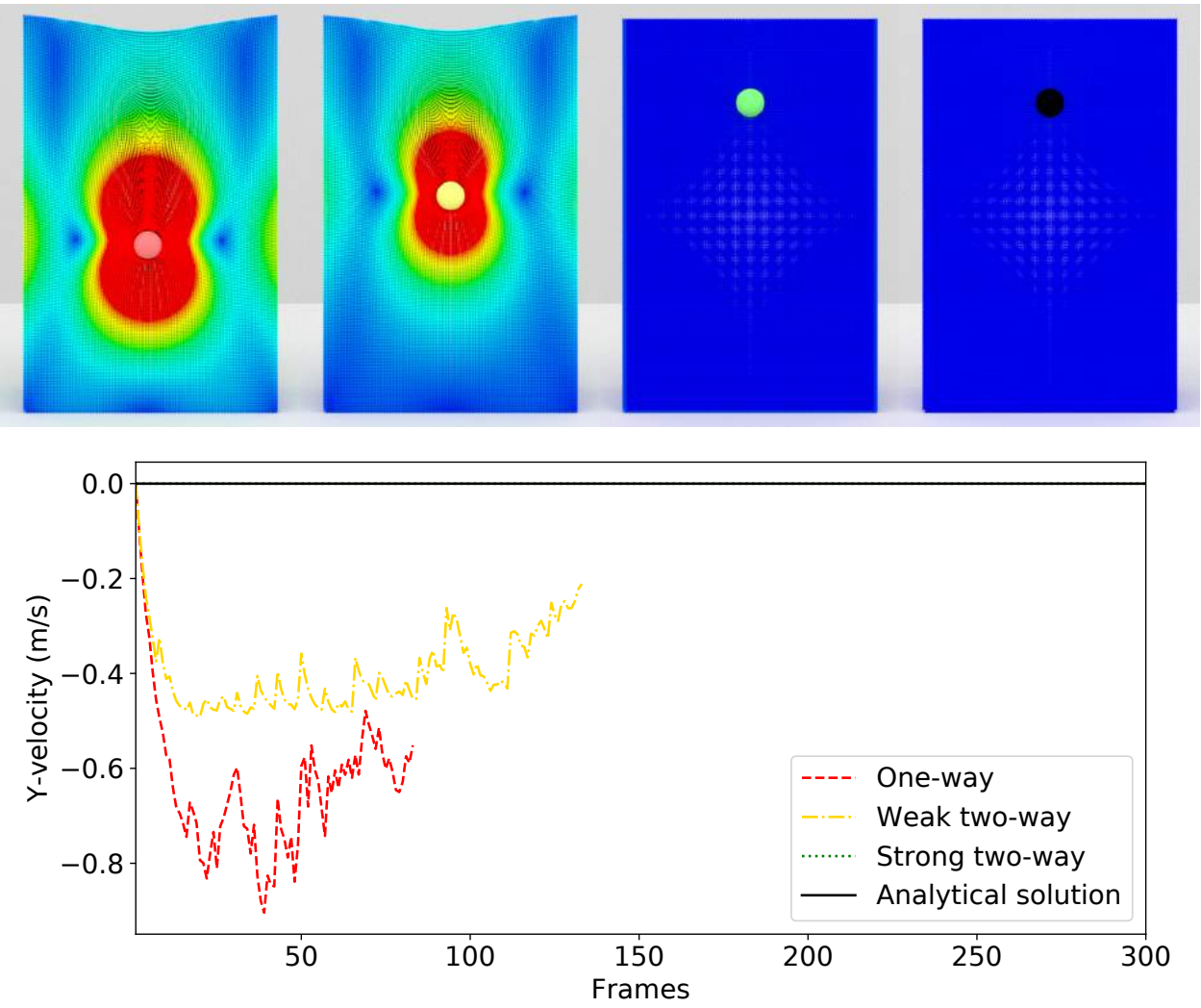


Figure D.5: A solid ball falling inside of fluids with viscosity values $\eta = 1.0 \times 10^5\ \mathrm{kg}/(\mathrm{s} \cdot \mathrm{m})$. (Top) From left to right, one-way coupling, weak two-way coupling, and strong two-way coupling. (Bottom) Profile of the y-directional velocity of the solid balls. My strong two-way coupling gives solid velocities very close to the analytical solution while the solid velocities given with one-way and weak two-way coupling significantly deviate from the analytical solution.

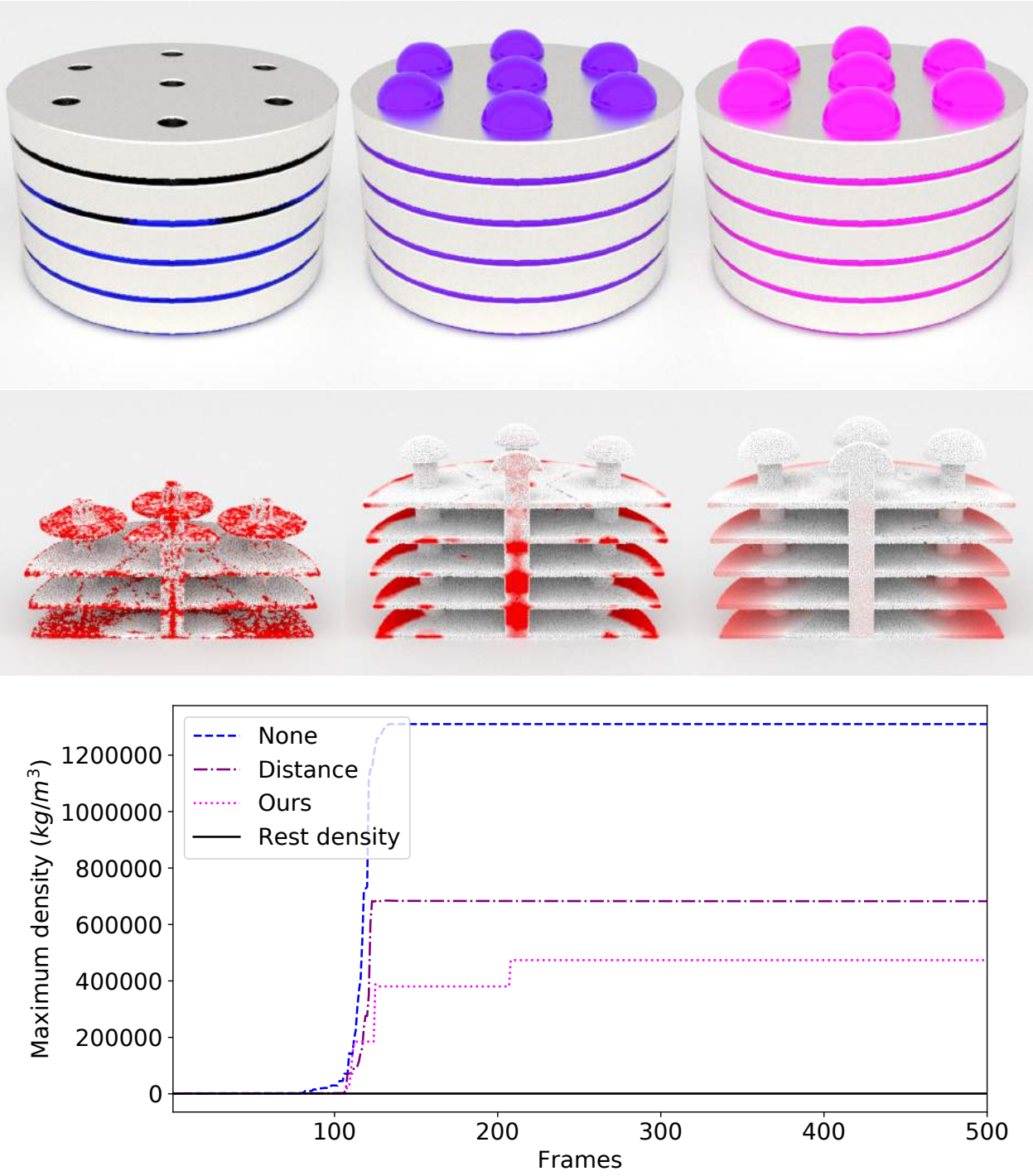


Figure D.6: (Top) A viscous fluid volume successively compressed by prescribed circular plates with several holes. From left to right, no position correction, distance-based position correction, and my method for surface rendering and particle view with color coding (white and red represent low and high densities, respectively). (Bottom) Profile of the maximum particle density, which indicates the inverse of local volumes. Compared to other approaches, my method preserves the density closer to the original one.

D.5 Implementation Details on J

Given viscous stress $\mathbf{s} = (s_{xx}, s_{xy}, s_{xz}, s_{yy}, s_{yz}, s_{zz})^T$ defined on the staggered grid, the translational viscosity forces (F_x, F_y, F_z) applied from fluids to a rigid body can be written with grid indices i, j, k, control volumes V and voxel size Δx as

$$F_x = \sum_{i,j,k} \big(J_{xx,i,j,k} s_{xx,i,j,k} + J_{xy,i-1/2,j-1/2,k} s_{xy,i-1/2,j-1/2,k} + J_{xz,i-1/2,j,k-1/2} s_{xz,i-1/2,j,k-1/2}\big),$$
$$F_y = \sum_{i,j,k} \big(J_{yx,i-1/2,j-1/2,k} s_{xy,i-1/2,j-1/2,k} + J_{yy,i,j,k} s_{yy,i,j,k} + J_{yz,i,j-1/2,k-1/2} s_{yz,i,j-1/2,k-1/2}\big),$$
$$F_z = \sum_{i,j,k} \big(J_{zx,i-1/2,j,k-1/2} s_{zx,i-1/2,j,k-1/2} + J_{zy,i,j-1/2,k-1/2} s_{yz,i,j-1/2,k-1/2} + J_{zz,i,j,k} s_{zz,i,j,k}\big),$$

where

$$J_{xx,i,j,k} = \frac{-V_{i+1/2,j,k} + V_{i-1/2,j,k}}{\Delta x},$$
$$J_{xy,i-1/2,j-1/2,k} = \frac{-V_{i-1/2,j,k} + V_{i-1/2,j-1,k}}{\Delta x},$$
$$J_{xz,i-1/2,j,k-1/2} = \frac{-V_{i-1/2,j,k} + V_{i-1/2,j,k-1}}{\Delta x},$$

$$J_{yx,i-1/2,j-1/2,k} = \frac{-V_{i,j-1/2,k} + V_{i-1,j-1/2,k}}{\Delta x},$$
$$J_{yy,i,j,k} = \frac{-V_{i,j+1/2,k} + V_{i,j-1/2,k}}{\Delta x},$$
$$J_{yz,i,j-1/2,k-1/2} = \frac{-V_{i,j-1/2,k} + V_{i,j-1/2,k-1}}{\Delta x},$$

$$J_{zx,i-1/2,j,k-1/2} = \frac{-V_{i,j,k-1/2} + V_{i-1,j,k-1/2}}{\Delta x},$$
$$J_{zy,i,j-1/2,k-1/2} = \frac{-V_{i,j,k-1/2} + V_{i,j-1,k-1/2}}{\Delta x},$$
$$J_{zz,i,j,k} = \frac{-V_{i,j,k+1/2} + V_{i,j,k-1/2}}{\Delta x}.$$

Ignoring the grid indices for readability (as of now), the rotational forces F_{rx}, F_{ry}, F_{rz} can be written with positions of viscous stress defined on a grid $\mathbf{x} = (x, y, z)^T$ and the center of mass for the rigid body

$\mathbf{X} = (X, Y, Z)^T$ as

$$
\begin{aligned}
F_{rx} &= \sum \left((y - Y)(J_{zx}s_{xz} + J_{zy}s_{yz} + J_{zz}s_{zz}) - (z - Z)(J_{yx}s_{xy} + J_{yy}s_{yy} + J_{yz}s_{yz}) \right), \\
F_{ry} &= \sum \left((z - Z)(J_{xx}s_{xx} + J_{xy}s_{xy} + J_{xz}s_{xz}) - (x - X)(J_{zx}s_{xz} + J_{zy}s_{yz} + J_{zz}s_{zz}) \right), \\
F_{rz} &= \sum \left((x - X)(J_{yx}s_{xy} + J_{yy}s_{yy} + J_{yz}s_{yz}) - (y - Y)(J_{xx}s_{xx} + J_{xy}s_{xy} + J_{xz}s_{xz}) \right).
\end{aligned}
$$

Since the viscous stresses are defined at different locations on the grid in the staggered manner, in practice, I compute the rotational forces above, e.g., for F_{rz} by

$$
\begin{aligned}
F_{rz} = \sum_{i,j,k} \Big(& (x_{i,j,k} - X) J_{yy,i,j,k} s_{yy,i,j,k} - (y_{i,j,k} - Y) J_{xx,i,j,k} s_{xx,i,j,k} \\
& + \left((x_{i-1/2,j-1/2,k} - X) J_{yx,i-1/2,j-1/2,k} - (y_{i-1/2,j-1/2,k} - Y) J_{xy,i-1/2,j-1/2,k} \right) s_{xy,i-1/2,j-1/2,k} \\
& + (x_{i-1/2,j,k-1/2} - X) J_{yz,i,j-1/2,k-1/2} s_{yz,i,j-1/2,k-1/2} \\
& - (y_{i,j-1/2,k-1/2} - Y) J_{xz,i-1/2,j,k-1/2} s_{xz,i-1/2,j,k-1/2} \Big).
\end{aligned}
$$

Similarly, I can compute F_{rx} and F_{ry}. Given $\mathbf{F} = (F_x, F_y, F_z, F_{rx}, F_{ry}, F_{rz})^T$ as the generalized six-dimensional viscosity forces for a rigid body, by extracting coefficients for the viscosity forces to assemble $\mathbf{J}$, I obtain the following relation:

$$
\mathbf{F} = \mathbf{J}\mathbf{s}.
$$

APPENDIX E: VISCOSITY PARAMETER IDENTIFICATION WITH PIV

E.1 Introduction

In this appendix, I describe the viscosity parameter identification with particle image velocimetry (PIV) data for viscous fluids captured from real-world experiments. In the experiments, I use a simple setup, where a solid ball falls inside of viscous fluids, to capture the velocity fields with PIV because of the ease of the experiments in the same condition without a special setup. Although it is possible to identify viscosity parameters with a simpler way, e.g., by matching the velocity of the solid ball with a simulated solid ball, in this scenario, I note that the simple approach is not applicable if the solid ball is moved in a prescribed manner while PIV data can still be used for parameter identification, and thus can be considered as more general. In this report, I explain the parameter identification with PIV data and provide some experimental results as an extra validation for my general parameter identification framework.

The algorithm of my framework with PIV data is essentially same as the one for example videos, and my goal is to identify viscosity parameters with which my viscous fluid simulator generates fluid flows as close as possible to the PIV data. Similar to the case of example videos, my framework first captures velocity fields using PIV from real world fluid phenomena, and preprocesses the captured data to make them amenable for the optimization. Then, I perform iterative optimization with forward viscous fluid simulations and finally output identified viscosity parameters. In the following, I describe major processes for PIV data, i.e., objective function formulation §E.1.1, velocity field capture §E.1.2, preprocess §E.1.3, and objective function evaluation §E.1.4. The iterative optimization can be performed in the same way as for the identification with example videos.

E.1.1 Objective Function

PIV is an optical method for directly capturing the velocity fields of fluid flows in the real world, and is widely used in the scientific fields for validation purposes. While there are various types of PIV setup and related algorithm, e.g., to capture 3D velocity fields (Xiong et al., 2017), one commonly available PIV system captures 2D velocity fields on a laser sheet injected by the system, as shown in Figure E.1.

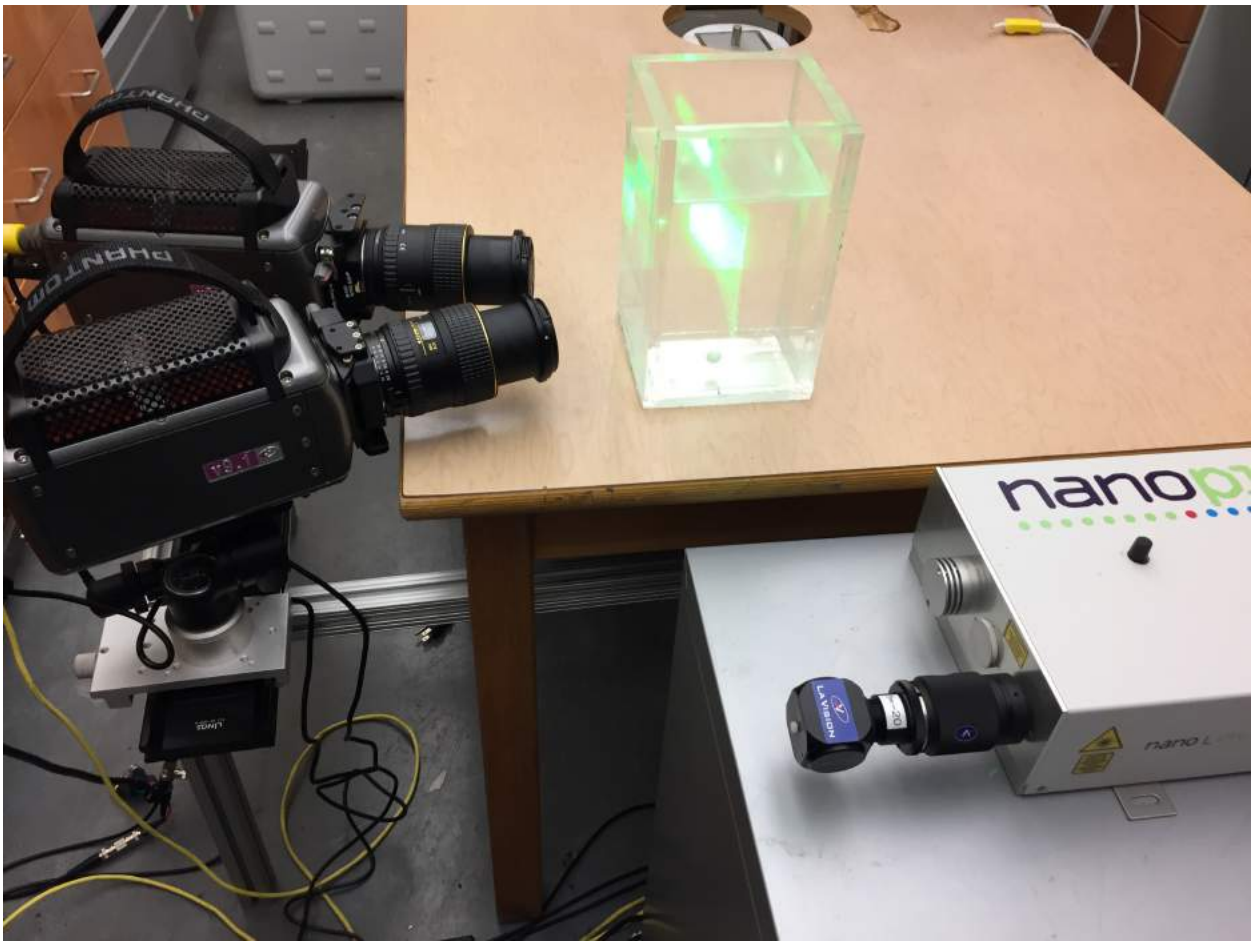


Figure E.1: My PIV setup to capture velocity fields. Laser is injected into viscous fluids to capture the 2D velocity fields on the sheet.

Since PIV can directly measure the fluid velocities from real fluid flows, I formulate my objective function as

$$E = \frac{1}{M} \sum_{f=0}^{N-1} \|\mathbf{C}_f^{\frac{1}{2}}(\tilde{\mathbf{u}}_f - \hat{\mathbf{u}}_f)\|_2^2, \tag{E.1}$$

where M denotes the number of velocity samples used in the optimization, f index for frames, N total count of frames considered in the optimization, $\mathbf{C}$ a diagonal coefficient matrix, $\tilde{\mathbf{u}}$ the 2-dimensional interpolated fluid velocities from the simulation, and $\hat{\mathbf{u}}$ the 2-dimensional fluid velocities captured with PIV.

E.1.2 Capturing Velocity Fields

To capture the velocity fields using PIV, I setup an experimental setting, as shown in Figure E.1. In my setting, I first prepare a container filled with viscous fluids, two calibrated and synchronized cameras positioned next to each other, and a laser device to inject thin sheet-shaped laser into the viscous fluids. Then, I put tiny metal particles into the viscous fluids so that these particles reflect the injected laser, and the cameras can capture the movement and velocities of these particles using optical flow algorithms. The resulting velocity fields computed with PIV are uniformly aligned and within a small window on the 2D (xy) plane produced by the injected laser.

To induce fluid velocities, I chose a simple scenario, where a solid ball falls down inside of the viscous fluids, so that one can easily and consistently regenerate a similar setup (see Figure E.3 (left)). I carefully put the solid ball such that the center of the solid ball is exactly on the sheet created by the laser device not to induce out of plane velocities (i.e., z-component of fluid velocities equals 0), and then measure velocity fields perturbed due to the falling solid ball. One example of the 2D velocity fields captured with PIV is shown in Figure E.2 (top left).

E.1.3 Preprocessing

While the velocity fields taken with PIV can sufficiently capture an overall flow of viscous fluids, there are some inconsistency due to the noise, which can be interpreted as unnatural, sudden velocity changes. Thus, I aim to remove the inconsistency from the captured velocity fields to make them temporally consistent and amenable in the optimization step.

Since the captured velocity fields are taken from the real fluid flows, their behaviors should follow the Newton's law, and thus the velocity fields should be sufficiently smooth in the temporal direction. Thus, I first measure the smoothness of the captured velocity fields with the Laplacian on the temporal direction:

$$\nabla_f^2 \hat{\mathbf{u}} = \frac{\hat{\mathbf{u}}_{f+1} - 2\hat{\mathbf{u}}_f + \hat{\mathbf{u}}_{f-1}}{\Delta t^2}. \tag{E.2}$$

Then, I evaluate the validity of the captured velocities by comparing the magnitude of the Laplacian $\|\nabla_f^2 \hat{\mathbf{u}}\|$ with the norm of the measured velocity itself $\|\hat{\mathbf{u}}\|$, and treat velocity fields as valid if $\|\nabla_f^2 \hat{\mathbf{u}}\| < \alpha \|\hat{\mathbf{u}}\|$, where α denotes a threshold parameter to adjust the necessary smoothness to be valid. In practice, I eliminate

invalid velocities by setting coefficients in the objective function as

$$c_f = \begin{cases} 1 & \text{if } \|\nabla_f^2 \hat{\mathbf{u}}\| < \alpha \|\hat{\mathbf{u}}\| \\ 0 & \text{otherwise} \end{cases} \tag{E.3}$$

where c denotes a diagonal entry of the coefficient matrix. It is worth noting that although non-binary coefficients could be used, I found that binary coefficients are generally preferable because the binary coefficients can completely eliminate the noisy velocity fields, and in general there are sufficient numbers of valid velocity fields to be used as a reference. Figure E.2 (top right) illustrates only valid velocity fields.

E.1.4 Evaluating Objective Function

To evaluate the objective function, it is necessary to perform the fluid simulation. For the simulation setup, I measure the size of the container and solid ball and density of the ball. Then, I manually estimate positions of the ball and compute solid velocities from the positions using finite difference. During the simulation, I interpolate the fluid velocities at the positions, where valid velocities are defined (see Figure E.2 (bottom left)). Since the velocity fields from the fluid simulation are available over the entire simulation domain (unlike PIV velocity fields), I can straightforwardly interpolate the velocities and compute the objective function. The difference between the valid PIV data and the interpolated velocities is illustrated in Figure E.2 (bottom right). I note that since fluid velocities are extrapolated into the solid ball, the computation of the objective function is valid even if the positions of the solid ball deviate from those in the example data.

In my framework, velocity fields obtained from PIV are available only within a small window on the 2D sheet. As such, it is not possible to replace the PIV velocity fields as initial or intermediate velocity fields for each step of the forward simulation. Consequently, it is necessary to rely on the simulation results at each frame, which would deviate from the example data due to the accumulated errors over multiple steps. However, I minimize the velocity deviations from the example data over multiple frames as a space-time optimization problem, and thus the resulting viscosity parameters are considered as optimal over the given time span although different parameters would generate smaller velocity deviations from the example data for a specific short term.

Table E.1: Viscosity parameter identification results with PIV velocity fields. ρ_f denotes fluid density (kg/m^3), $\hat{\eta}$ fluid viscosity $(kg/(s \cdot m))$, r solid ball radius $1.0 \times 10^{-3}(m)$, ρ_s solid ball density (kg/m^3), u_∞ the terminal velocity of the solid ball (m/s), Re Reynolds number, η identified viscosity value $(kg/(s \cdot m))$, and ϵ relative error (%). In general, relative errors are small.

Name	ρ_f	$\hat{\eta}$	r	ρ_s	u_∞	Re	η	ϵ
plastic	970.79	1.03	6.25	1555.00	-0.04	0.47	0.96	6.80
steel	970.79	1.03	4.76	8050.00	-0.12	1.07	1.12	8.73

E.2 Validation Results

I implemented my framework with C++, and used a viscous fluid solver based on (Batty and Bridson, 2008; Takahashi and Lin, 2019a) My experiments are executed on a Linux machine with 24-core 2.50GHz Intel Xeon and 256 GB RAMs. For the parameter identification, I typically formulate the objective function with up to 50 frames to make the space-time optimization manageable.

E.2.1 PIV Velocity Fields

To validate my framework, I use a simple experimental setting, where a solid ball is falling inside of viscous fluids. This experimental setting is shown in Figure E.3 (left). I capture velocity fields with this setup, and used the captured velocity fields as input for my framework. In this experiment, I use silicone oils as viscous fluids and measured their viscosity values with a viscometer for comparison. I use different types of balls: "plastic" and "steel". Because of different size and density of the balls, the falling speed of the balls are also different leading to distinct fluid flow patterns (i.e., different Reynolds numbers). The parameters and results are summarized in Table E.1. I note that in the scenes "plastic" and "steel", Stokes' law is not valid since Reynolds number is not sufficiently low ($Re \ll 1$), and thus it is not possible to identify viscosity parameters based on the Stokes' law. In this experiment, for the identification results of "plastic" and "steel", the relative errors are small and is within 10%.

For the demonstration of the parameter identification result, I simulate the falling sphere scenario, where I captured the PIV velocity fields. The result is given in Figure E.3 (right) and in the accompanying video. The resulting movement of the simulated solid ball is in good agreement with the ball in the real world counterpart.

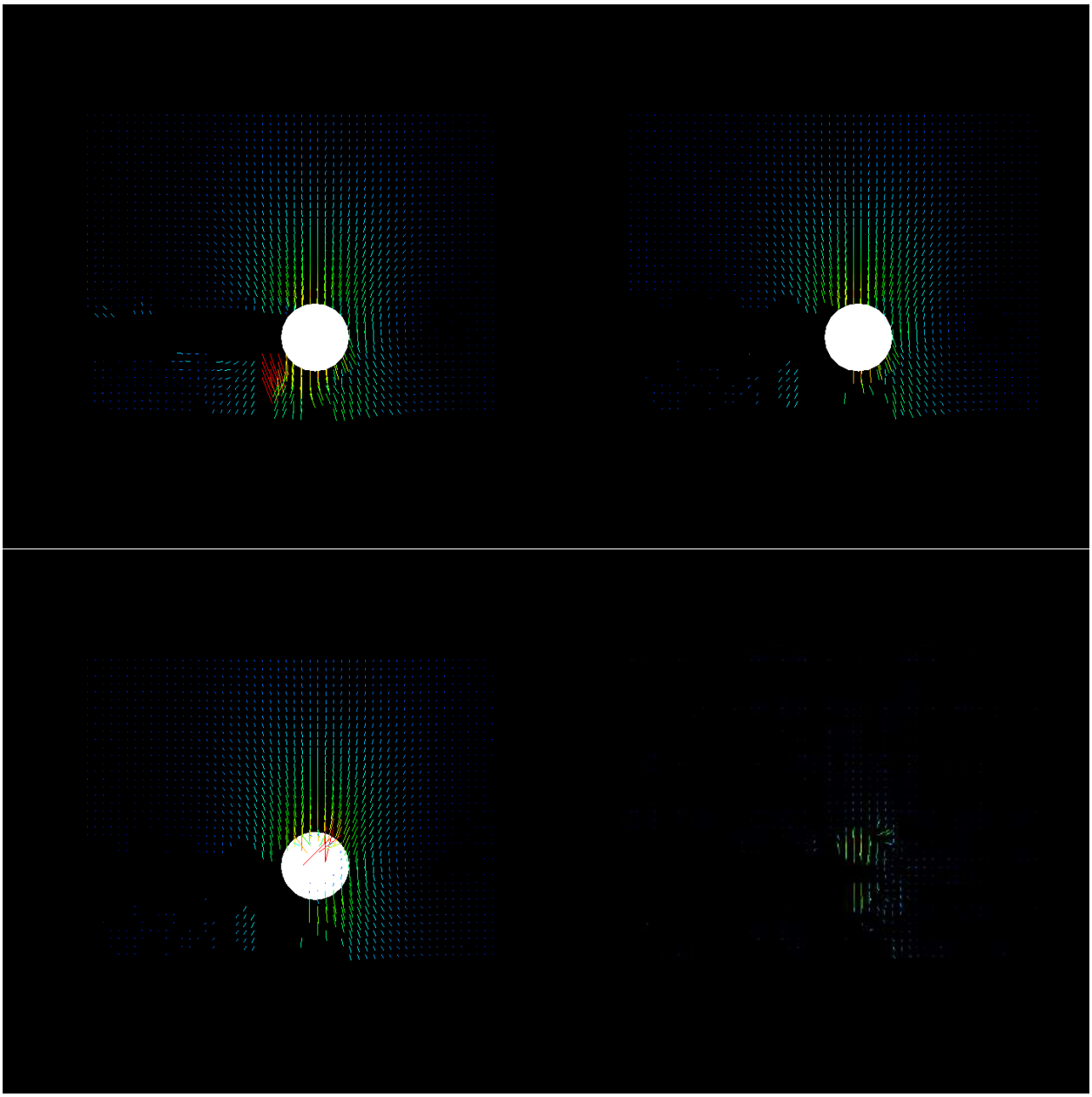


Figure E.2: Velocity field illustration. White sphere represents the solid ball falling inside the viscous fluids. (Top left) velocity fields captured with PIV. (Top right) valid velocity fields. (Bottom left) velocity fields interpolated from velocity fields generated with my solver. (Bottom right) Velocity field differences between the valid velocity fields and the interpolated velocity fields.

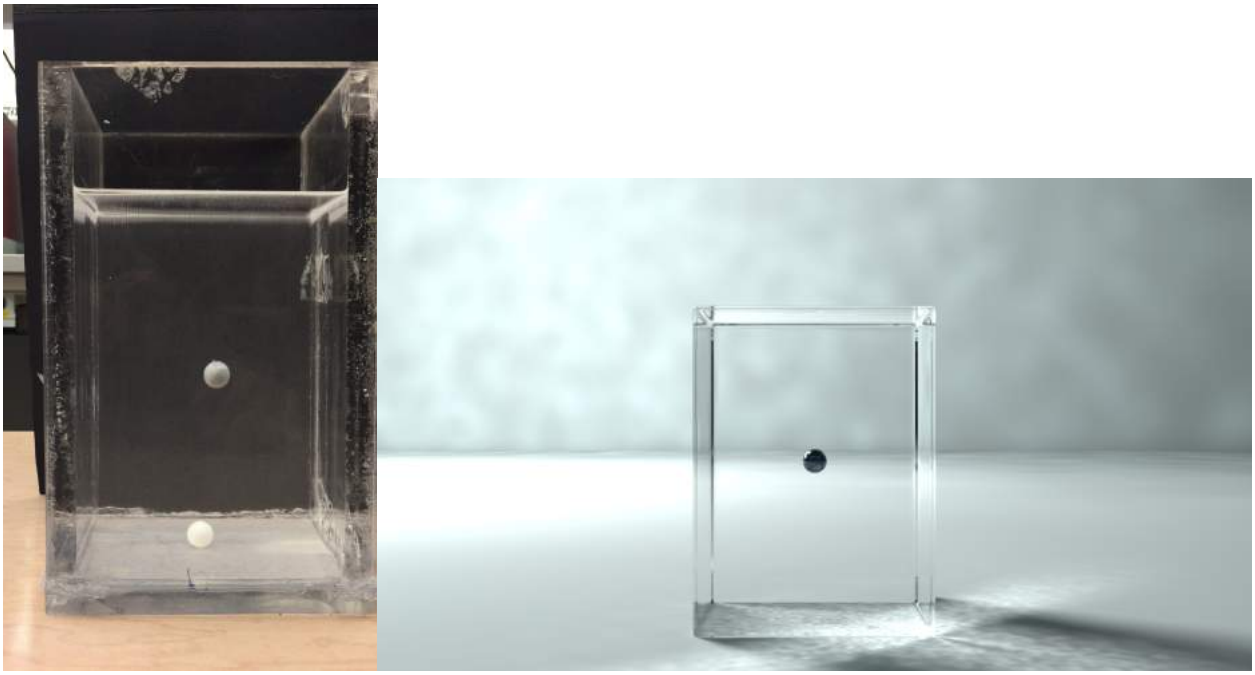


Figure E.3: A sphere falling inside of viscous fluids. (Left) experimental setup for capturing PIV velocity fields. (Right) my simulation results with the identified viscosity parameter. The falling behavior of the simulated ball is in good agreement with the falling ball in the real world used to capture PIV data.

BIBLIOGRAPHY

Aanjaneya, M. (2018). An efficient solver for two-way coupling rigid bodies with incompressible flow. *Computer Graphics Forum*, 37(8).

Adams, B., Pauly, M., Keiser, R., and Guibas, L. J. (2007). Adaptively sampled particle fluids. *ACM Transactions on Graphics*, 26(3).

Adams, B. and Wicke, M. (2009). Meshless approximation methods and applications in physics based modeling and animation. In *Eurographics 2009 Tutorials*, pages 213–239.

Akbay, M., Nobles, N., Zordan, V., and Shinar, T. (2018). An extended partitioned method for conservative solid-fluid coupling. *ACM Trans. Graph.*

Akinci, N., Ihmsen, M., Akinci, G., Solenthaler, B., and Teschner, M. (2012). Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics*, 31(4):62:1–62:8.

Alduán, I. and Otaduy, M. A. (2011). SPH granular flow with friction and cohesion. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 25–32.

Ando, R., Thuerey, N., and Wojtan, C. (2015). A stream function solver for liquid simulations. *ACM Trans. Graph.*, 34(4):53:1–53:9.

Ando, R. and Tsuruno, R. (2011). A particle-based method for preserving fluid sheets. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–16.

Andrade, Luiz, F. d. S., Sandim, M., Petronetto, F., Pagliosa, P., and Paiva, A. (2014). SPH fluids for viscous jet buckling. In *Proceedings of SIBGRAPI 2014*, pages 65–72.

Atcheson, B., Ihrke, I., Heidrich, W., Tevs, A., Bradley, D., Magnor, M., and Seidel, H.-P. (2008). Time-resolved 3d capture of non-stationary gas flows. *ACM Trans. Graph.*, 27(5):132:1–132:9.

Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G. (2007). A finite element method for animating large viscoplastic flow. *ACM Transactions on Graphics*, 26(3).

Barreiro, H., García-Fernández, I., Alduán, I., and Otaduy, M. A. (2017). Conformation constraints for efficient viscoelastic fluid simulation. *ACM Trans. Graph.*, 36(6):221:1–221:11.

Batty, C. (2008). Variationalviscosity3D https://github.com/christopherbatty/variationalviscosity3d.

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

Batty, C. and Bridson, R. (2008). Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 219–228.

Batty, C. and Houston, B. (2011). A simple finite volume method for adaptive viscous liquids. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 111–118.

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

Becker, M. and Teschner, M. (2007a). Robust and efficient estimation of elasticity parameters using the linear finite element method. In *Simulation and Visualization*, pages 15–28.

Becker, M. and Teschner, M. (2007b). Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 209–217.

Becker, M., Tessendorf, H., and Teschner, M. (2009). Direct forcing for lagrangian rigid-fluid coupling. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):493–503.

Bender, J. and Koschier, D. (2015). Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 2015 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Bender, J. and Koschier, D. (2016). Divergence-free sph for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1.

Bergou, M., Audoly, B., Vouga, E., Wardetzky, M., and Grinspun, E. (2010). Discrete viscous threads. *ACM Transactions on Graphics*, 29(4):116:1–116:10.

Bhat, K. S., Seitz, S. M., and Popovic, J. (2002). Computing the physical parameters of rigid-body motion from video. In *Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 551–565.

Bhat, K. S., Twigg, C. D., Hodgins, J. K., Khosla, P. K., Popović, Z., and Seitz, S. M. (2003). Estimating cloth simulation parameters from video. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 37–51.

Bickel, B., Bächer, M., Otaduy, M. A., Lee, H. R., Pfister, H., Gross, M., and Matusik, W. (2010). Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.*, 29(4):63:1–63:10.

Bickel, B., Bächer, M., Otaduy, M. A., Matusik, W., Pfister, H., and Gross, M. (2009). Capture and modeling of non-linear heterogeneous soft tissue. *ACM Trans. Graph.*, 28(3):89:1–89:9.

Bodin, K., Lacoursiere, C., and Servin, M. (2012). Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):516–526.

Bridson, R. (2008). *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press.

Bridson, R. (2015). *Fluid Simulation for Computer Graphics*. A K Peters/CRC Press.

Briggs, W., Henson, V., and Mccormick, S. (2000). *A Multigrid Tutorial, Second Edition*. Society for Industrial and Applied Mathematics.

Carlson, M., Mucha, P. J., and Turk, G. (2004). Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3):377–384.

Carlson, M., Mucha, P. J., Van Horn, III, R. B., and Turk, G. (2002). Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 167–174.

Chen, D., Li, W., and Hall, P. (2016). Dense motion estimation for smoke. In *Asian Conference on Computer Vision*, Asian Conference on Computer Vision.

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

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

Chentanez, N. and Müller, M. (2012). A multigrid fluid pressure solver handling separating solid boundary conditions. *Visualization and Computer Graphics, IEEE Transactions on*, 18(8):1191 –1201.

Clausen, P., Wicke, M., Shewchuk, J. R., and O'Brien, J. F. (2013). Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Transactions on Graphics*, 32(2):17:1–17:15.

Clavet, S., Beaudoin, P., and Poulin, P. (2005). Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 219–228.

Clyde, D., Teran, J., and Tamstorf, R. (2017). Modeling and data-driven parameter estimation for woven fabrics. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 17:1–17:11.

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

Corpetti, T., Memin, E., and Perez, P. (2002). Dense estimation of fluid flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):365–380.

Cummins, S. J. and Rudman, M. (1999). An SPH projection method. *Journal of Computational Physics*, 152(2):584–607.

Dagenais, F., Gagnon, J., and Paquette, E. (2012). A prediction-correction approach for stable SPH fluid simulation from liquid to rigid. In *Proceedings of the Computer Graphics International 2012*.

Daviet, G. and Bertails-Descoubes, F. (2016). A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans. Graph.*, 35(4):102:1–102:13.

de Goes, F., Wallez, C., Huang, J., Pavlov, D., and Desbrun, M. (2015). Power particles: An incompressible fluid solver based on power diagrams. *ACM Trans. Graph.*, 34(4):50:1–50:11.

Derouet-Jourdan, A., Bertails-Descoubes, F., Daviet, G., and Thollot, J. (2013). Inverse dynamic hair modeling with frictional contact. *ACM Trans. Graph.*, 32(6):159:1–159:10.

Desbrun, M. and Gascuel, M.-P. (1996). Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 61–76.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. (1999). Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 317–324.

Dick, C., Rogowsky, M., and Westermann, R. (2016). Solving the fluid pressure poisson equation using multigrid–evaluation and improvements. *Visualization and Computer Graphics, IEEE Transactions on*.

Dostal, Z. and Schoberl, J. (2005). Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. *Computational Optimization and Applications*, 30(1):23–43.

Eckert, M.-L., Heidrich, W., and Thürey, N. (2018). Coupled fluid density and motion from single views. *Comput. Graph. Forum*, 37(8):47–58.

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

Fan, X.-J., Tanner, R., and Zheng, R. (2010). Smoothed particle hydrodynamics simulation of non-Newtonian moulding flow. *Journal of Non-Newtonian Fluid Mechanics*, 165(5-6):219–226.

Fang, Y., Li, M., Gao, M., and Jiang, C. (2019). Silly rubber: An implicit material point method for simulating non-equilibrated viscoelastic and elastoplastic solids. *ACM Trans. Graph.*

Feldman, B. E., O'Brien, J. F., and Arikan, O. (2003). Animating suspended particle explosions. *ACM Trans. Graph.*, 22(3):708–715.

Ferstl, F., Westermann, R., and Dick, C. (2014). Large-scale liquid simulation on adaptive hexahedral grids. *Visualization and Computer Graphics, IEEE Transactions on*, 20(10):1405–1417.

Gao, M., Pradhana, A., Han, X., Guo, Q., Kot, G., Sifakis, E., and Jiang, C. (2018). Animating fluid sediment mixture in particle-laden flows. *ACM Trans. Graph.*

Gao, M., Tampubolon, A. P., Jiang, C., and Sifakis, E. (2017). An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Trans. Graph.*, 36(6):223:1–223:12.

Gerlach, S. and Matzenmiller, A. (2007). On parameter identification for material and microstructural properties. *GAMM-Mitteilungen*, 30(2):481–505.

Gerszewski, D. and Bargteil, A. W. (2013). Physics-based animation of large-scale splashing liquids. *ACM Transactions on Graphics*, 32(6):185:1–185:6.

Gerszewski, D., Bhattacharya, H., and Bargteil, A. W. (2009). A point-based method for animating elastoplastic solids. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 133–138.

Gibou, F., Fedkiw, R. P., Cheng, L.-T., and Kang, M. (2002). A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205 – 227.

Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F. (2004). A method for animating viscoelastic fluids. *ACM Trans. Graph.*, 23(3):463–468.

Goldade, R., Wang, Y., Aanjaneya, M., and Batty, C. (2019). An adaptive variational finite difference framework for efficient symmetric octree viscosity. *ACM Trans. Graph.*

Grant, I. (1997). Particle image velocimetry: a review. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 211:55–76.

Gregson, J., Ihrke, I., Thuerey, N., and Heidrich, W. (2014). From capture to simulation: Connecting forward and inverse problems in fluids. *ACM Trans. Graph.*, 33(4):139:1–139:11.

Gregson, J., Krimerman, M., Hullin, M. B., and Heidrich, W. (2012). Stochastic tomography and its applications in 3d imaging of mixing fluids. *ACM Trans. Graph.*, 31(4):52:1–52:10.

Guendelman, E., Selle, A., Losasso, F., and Fedkiw, R. (2005). Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph.*, 24(3):973–981.

Hasinoff, S. W. and Kutulakos, K. N. (2007). Photo-consistent reconstruction of semitransparent scenes by density-sheet decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):870–885.

He, X., Liu, N., Li, S., Wang, H., and Wang, G. (2012a). Local poisson SPH for viscous incompressible fluids. *Computer Graphics Forum*, 31(6):1948–1958.

He, X., Liu, N., Wang, G., Zhang, F., Li, S., Shao, S., and Wang, H. (2012b). Staggered meshless solid-fluid coupling. *ACM Transactions on Graphics*, 31(6):149:1–149:12.

He, X., Wang, H., Zhang, F., Wang, H., Wang, G., and Zhou, K. (2014). Robust simulation of sparsely sampled thin features in SPH-based free surface flows. *ACM Transactions on Graphics*, 34(1):7:1–7:9.

Hu, L., Bradley, D., Li, H., and Beeler, T. (2017). Simulation-ready hair capture. *Comput. Graph. Forum*, 36:281–294.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., and Jiang, C. (2018). A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Trans. Graph.*

Ihmsen, M. (2013). *Particle-based Simulation of Large Bodies of Water with Bubbles, Spray and Foam*. PhD thesis, University of Freiburg.

Ihmsen, M., Akinci, N., Becker, M., and Teschner, M. (2011). A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum*, 30(1):99–112.

Ihmsen, M., Cornelis, J., Solenthaler, B., Horvath, C., and Teschner, M. (2014a). Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435.

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., and Teschner, M. (2014b). SPH fluids in computer graphics. In *EUROGRAPHICS 2014 State of the Art Reports*, pages 21–42.

Ihmsen, M., Wahl, A., and Teschner, M. (2013). A Lagrangian framework for simulating granular material with high detail. *Computers & Graphics*, 37(7):800–808.

Ihrke, I. and Magnor, M. (2004). Image-based tomographic reconstruction of flames. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 365–373.

Jiang, C., Gast, T., and Teran, J. (2017). Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans. Graph.*, 36(4):152:1–152:14.

Jiang, C., Schroeder, C., Selle, A., Teran, J., and Stomakhin, A. (2015). The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4):51:1–51:10.

Jones, B., Ward, S., Jallepalli, A., Perenia, J., and Bargteil, A. W. (2014). Deformation embedding for point-based elastoplastic simulation. *ACM Transactions on Graphics*, 33(2):21:1–21:9.

Kang, N. and Sagong, D. (2014). Incompressible sph using the divergence-free condition. *Computer Graphics Forum*, 33(7):219–228.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M. H., and Solenthaler, B. (2019). Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*.

Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J. (2007). Simulation of bubbles in foam with the volume control method. *ACM Trans. Graph.*, 26(3).

Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C., and Teran, J. (2016). Drucker-prager elastoplasticity for sand animation. *ACM Trans. Graph.*, 35(4):103:1–103:12.

Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F. (2006). Fluid animation with dynamic meshes. *ACM Trans. Graph.*, 25(3).

Koshizuka, S., Tamako, H., and Oka, Y. (1996). A particle method for incompressible viscous flow with fluid fragmentations. *Computational Fluid Dynamics Journal*, 4(1):29–46.

Laibe, G. and Price, D. J. (2012). Dusty gas with smoothed particle hydrodynamics - II. implicit timestepping and astrophysical drag regimes. *Monthly Notices of the Royal Astronomical Society*, 420(3).

Larionov, E., Batty, C., and Bridson, R. (2017). Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4):101:1–101:11.

Lee, H.-P. and Lin, M. C. (2012). Fast optimization-based elasticity parameter estimation using reduced models. *The Visual Computer*, 28(6):553–562.

Lentine, M., Zheng, W., and Fedkiw, R. (2010). A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph.*, 29(4):114:1–114:9.

Li, C., Pickup, D., Saunders, T., Cosker, D., Marshall, D., Hall, P., and Willis, P. (2013). Water surface modeling from a single viewpoint video. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1242–1251.

Li, S., Huang, J., de Goes, F., Jin, X., Bao, H., and Desbrun, M. (2014). Space-time editing of elastic motion through material optimization and reduction. *ACM Trans. Graph.*, 33(4):108:1–108:10.

Lu, W., Jin, N., and Fedkiw, R. (2016). Two-way coupling of fluids to reduced deformable bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–76.

Ly, M., Casati, R., Bertails-Descoubes, F., Skouras, M., and Boissieux, L. (2018). Inverse elastic shell design with contact and friction. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 201:1–201:16.

Macklin, M. and Müller, M. (2013). Position based fluids. *ACM Transactions on Graphics*, 32(4):104:1–104:5.

Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y. (2014). Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 33(4):153:1–153:12.

McAdams, A., Sifakis, E., and Teran, J. (2010). A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74.

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. (2011). Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30(4):37:1–37:12.

McNamara, A., Treuille, A., Popović, Z., and Stam, J. (2004). Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456.

Miguel, E., Bradley, D., Thomaszewski, B., Bickel, B., Matusik, W., Otaduy, M. A., and Marschner, S. (2012). Data-driven estimation of cloth simulation models. *Comput. Graph. Forum*, 31(2pt2):519–528.

Miller, G. and Pearce, A. (1989). Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305–309.

Min, C. and Gibou, F. (2007). Geometric integration over irregular domains with application to level-set methods. *Journal of Computational Physics*, 226(2):1432 – 1443.

Monaghan, J. (1994). Simulating free surface flows with SPH. *J. Comput. Phys.*, 110(2):399–406.

Monaghan, J. (2000). SPH without a tensile instability. *Journal of Computational Physics*, 159(2):290 – 311.

Monaghan, J., J. (2005). Smoothed particle hydrodynamics. *Reports on Progress in Physics*.

Monaghan, J. J. (1989). On the problem of penetration in particle methods. *Journal of Computational Physics*, 82(1):1–15.

Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30:543–574.

Monaghan, J. J. (1997). Implicit SPH drag and dusty gas dynamics. *Journal of Computational Physics*, 138(2):801–820.

Monszpart, A., Thuerey, N., and Mitra, N. J. (2016). Smash: Physics-guided reconstruction of collisions from videos. *ACM Trans. Graph.*, 35(6):199:1–199:14.

Morris, N. J. W. and Kutulakos, K. N. (2011). Dynamic refraction stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1518–1531.

Müller, M. (2008). Hierarchical position based dynamics. In *VRIPHYS*, pages 1–10.

Müller, M., Charypar, D., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication Image Representation*, 18(2):109–118.

Müller, M., Solenthaler, B., Keiser, R., and Gross, M. (2005). Particle-based fluid-fluid interaction. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 237–244.

Nagasawa, K., Suzuki, T., Seto, R., Okada, M., and Yue, Y. (2019). Mixing sauces: A viscosity blending model for shear thinning fluids. *ACM Trans. Graph.*, 38(4):95:1–17.

Nair, P. and Tomar, G. (2014). An improved free surface modeling for incompressible SPH. *Computers & Fluids*, 102(10):304 – 314.

Narain, R., Golas, A., and Lin, M. C. (2010). Free-flowing granular materials with two-way solid coupling. *ACM Transactions on Graphics*, 29(6):173:1–173:10.

Ng, Y. T., Min, C., and Gibou, F. (2009). An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807 – 8829.

Okabe, M., Dobashi, Y., Anjyo, K., and Onai, R. (2015). Fluid volume modeling from sparse multi-view images by appearance transfer. *ACM Trans. Graph.*, 34(4):93:1–93:10.

Orthmann, J. and Kolb, A. (2012). Temporal blending for adaptive SPH. *Computer Graphics Forum*, 31(8):2436–2449.

Pai, D. K., Doel, K. v. d., James, D. L., Lang, J., Lloyd, J. E., Richmond, J. L., and Yau, S. H. (2001). Scanning physical interaction behavior of 3d objects. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 87–96.

Pai, D. K., Rothwell, A., Wyder-Hodge, P., Wick, A., Fan, Y., Larionov, E., Harrison, D., Neog, D. R., and Shing, C. (2018). The human touch: Measuring contact with real human soft tissues. *ACM Trans. Graph.*, 37(4):58:1–58:12.

Paiva, A., Petronetto, F., Lewiner, T., and Tavares, G. (2006). Particle-based non-newtonian fluid animation for melting objects. In *Proceedings of SIBGRAPI 2006.*, pages 78–85.

Peer, A., Ihmsen, M., Cornelis, J., and Teschner, M. (2015). An implicit viscosity formulation for sph fluids. *ACM Trans. Graph.*, 34(4):114:1–114:10.

Peer, A. and Teschner, M. (2017). Prescribed velocity gradients for highly viscous sph fluids with vorticity diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 23(12):2656–2662.

Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. T. (2003). Particle-based simulation of fluids. 22(3):401–410.

Rafiee, A., Manzari, M., and Hosseini, M. (2007). An incompressible sph method for simulation of unsteady viscoelastic free-surface flows. *International Journal of Non-Linear Mechanics*, 42(10):1210 – 1223.

Rasmussen, N., Enright, D., Nguyen, D., Marino, S., Sumner, N., Geiger, W., Hoon, S., and Fedkiw, R. (2004). Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 193–202.

Raveendran, K., Wojtan, C., and Turk, G. (2011). Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 33–42.

Ren, B., Li, C., Yan, X., Lin, M. C., Bonet, J., and Hu, S.-M. (2014). Multiple-fluid SPH simulation using a mixture model. *ACM Transactions on Graphics*, 33(5):171:1–171:11.

Ren, Z., Yeh, H., and Lin, M. C. (2013). Example-guided physically based modal sound synthesis. *ACM Trans. Graph.*, 32(1):1:1–1:16.

Robinson-Mosher, A., English, R. E., and Fedkiw, R. (2009). Accurate tangential velocities for solid fluid coupling. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 227–236.

Robinson-Mosher, A., Schroeder, C., and Fedkiw, R. (2011). A symmetric positive definite formulation for monolithic fluid structure interaction. *J. Comput. Phys.*, 230(4):1547–1566.

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

Sacht, L., Vouga, E., and Jacobson, A. (2015). Nested cages. *ACM Trans. Graph.*, 34(6):170:1–170:14.

Sato, T., Wojtan, C., Thuerey, N., Igarashi, T., and Ando, R. (2018). Extended Narrow Band FLIP for Liquid Simulations. *Computer Graphics Forum*.

Schechter, H. and Bridson, R. (2012). Ghost SPH for animating water. *ACM Transactions on Graphics*, 31(4):61:1–61:8.

Shao, S. and Lo, E. Y. (2003). Incompressible SPH method for simulating newtonian and non-newtonian flows with a free surface. *Advances in Water Resources*, 26(7):787 – 800.

Sin, F., Bargteil, A. W., and Hodgins, J. K. (2009). A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 247–255.

Solenthaler, B. and Gross, M. (2011). Two-scale particle simulation. *ACM Transactions on Graphics*, 30(4):81:1–81:8.

Solenthaler, B. and Pajarola, R. (2008). Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–218.

Solenthaler, B. and Pajarola, R. (2009). Predictive-corrective incompressible SPH. *ACM Transactions on Graphics*, 28(3):40:1–40:6.

Solenthaler, B., Schläfli, J., and Pajarola, R. (2007). A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82.

Stam, J. (1999). Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. (2013). A material point method for snow simulation. *ACM Trans. Graph.*, 32(4):102:1–102:10.

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., and Selle, A. (2014). Augmented MPM for phase-change and varied materials. *ACM Transactions on Graphics*, 33(4):138:1–138:11.

Takahashi, T., Dobashi, Y., Fujishiro, I., Nishita, T., and Lin, M. C. (2015). Implicit formulation for SPH-based viscous fluids. *Computer Graphics Forum*, 34(2):493–502.

Takahashi, T., Dobashi, Y., Nishita, T., and Lin, M. C. (2016). An efficient hybrid incompressible sph solver with interface handling for boundary conditions. *Computer Graphics Forum*.

Takahashi, T. and Lin, M. C. (2016). A multilevel sph solver with unified solid boundary handling. *Computer Graphics Forum*, 35(7):517–526.

Takahashi, T. and Lin, M. C. (2019a). A geometrically consistent viscous fluid solver with two-way fluid-solid coupling. *Computer Graphics Forum*.

Takahashi, T. and Lin, M. C. (2019b). Video-guided real-to-virtual parameter transfer for viscous fluids. *ACM Transactions on Graphics*.

Takahashi, T., Nishita, T., and Fujishiro, I. (2014). Fast simulation of viscous fluids with elasticity and thermal conductivity using position-based dynamics. *Computers & Graphics*, 43:21–30.

Tampubolon, A. P., Gast, T., Klár, G., Fu, C., Teran, J., Jiang, C., and Museth, K. (2017). Multi-species simulation of porous sand and water mixtures. *ACM Trans. Graph.*, 36(4):105:1–105:11.

Tamstorf, R., Jones, T., and McCormick, S. F. (2015). Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34(6):245:1–245:13.

Teng, Y., Levin, D. I. W., and Kim, T. (2016). Eulerian solid-fluid coupling. *ACM Trans. Graph.*, 35(6):200:1–200:8.

Terzopoulos, D., Platt, J., and Fleischer, K. (1991). Heating and melting deformable models. *Journal of Visualization and Computer Animation*, 2:68–73.

Tonge, R., Benevolenski, F., and Voroshilov, A. (2012). Mass splitting for jitter-free parallel rigid body simulation. *ACM Trans. Graph.*, 31(4):105:1–105:8.

Trottenberg, U., Oosterlee, C. W., and Schuller, A. (2000). *Multigrid*. Academic press.

Twigg, C. D. and Kačić-Alesić, Z. (2011). Optimization for sag-free simulations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 225–236.

Um, K., Baek, S., and Han, J. (2014). Advanced hybrid particle-grid method with sub-grid particle correction. *Comput. Graph. Forum*, 33(7):209–218.

Um, K., Hu, X., and Thuerey, N. (2017). Perceptual evaluation of liquid simulation methods. *ACM Trans. Graph.*, 36(4):143:1–143:12.

Vantzos, O., Raz, S., and Ben-Chen, M. (2018). Real-time viscous thin films. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 281:1–281:10.

Vines, M., Houston, B., Lang, J., and Lee, W.-S. (2014). Vortical inviscid flows with two-way solid-fluid coupling. *IEEE Transactions on Visualization and Computer Graphics*, 20(2):303–315.

Wang, B., Wu, L., Yin, K., Ascher, U., Liu, L., and Huang, H. (2015). Deformation capture and modeling of soft objects. *ACM Trans. Graph.*, 34(4):94:1–94:12.

Wang, H., Liao, M., Zhang, Q., Yang, R., and Turk, G. (2009). Physically guided liquid surface modeling from videos. *ACM Trans. Graph.*, 28(3):90:1–90:11.

Wang, H., O'Brien, J., and Ramamoorthi, R. (2010). Multi-resolution isotropic strain limiting. *ACM Trans. Graph.*, 29(6):156:1–156:10.

Wang, H., O'Brien, J. F., and Ramamoorthi, R. (2011). Data-driven elastic models for cloth: Modeling and measurement. *ACM Trans. Graph.*, 30(4):71:1–71:12.

Wang, S. (2013). 3D volume calculation for the marching cubes algorithm in cartesian coordinates. *Arxiv*.

Weber, D., Mueller-Roemer, J., Stork, A., and Fellner, D. (2015). A cut-cell geometric multigrid poisson solver for fluid simulation. *Computer Graphics Forum*, 34(2):481–491.

Weiler, M., Koschier, D., Brand, M., and Bender, J. (2018a). A physically consistent implicit viscosity solver for sph fluids. *Computer Graphics Forum (Eurographics)*, 37(2).

Weiler, M., Koschier, D., Brand, M., and Bender, J. (2018b). A physically consistent implicit viscosity solver for sph fluids. *Computer Graphics Forum*, 37(2):145–155.

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

Wojtan, C. and Turk, G. (2008). Fast viscoelastic behavior with thin features. *ACM Transactions on Graphics*, 27(3):47:1–47:8.

Xiong, J., Idoughi, R., Aguirre-Pablo, A. A., Aljedaani, A. B., Dun, X., Fu, Q., Thoroddsen, S. T., and Heidrich, W. (2017). Rainbow particle imaging velocimetry for dense 3d fluid velocity imaging. *ACM Trans. Graph.*, 36(4):36:1–36:14.

Xu, H. and Barbič, J. (2017). Example-based damping design. *ACM Trans. Graph.*, 36(4):53:1–53:14.

Xu, H., Li, Y., Chen, Y., and Barbič, J. (2015). Interactive material design using model reduction. *ACM Trans. Graph.*, 34(2):18:1–18:14.

Yan, G., Li, W., Yang, R., and Wang, H. (2018). Inexact descent methods for elastic parameter optimization. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 253:1–253:14.

Yang, S., Ambert, T., Pan, Z., Wang, K., Yu, L., Berg, T., and Lin, M. C. (2016). Detailed garment recovery from a single-view image.

Yang, S., Liang, J., and Lin, M. C. (2017). Learning-based cloth material recovery from video. In *2017 IEEE International Conference on Computer Vision*.

Yang, S. and Lin, M. C. (2016). Materialcloning: Acquiring elasticity parameters from images for medical applications. *IEEE Transactions on Visualization and Computer Graphics*, 22(9):2122–2135.

Yue, Y., Smith, B., Batty, C., Zheng, C., and Grinspun, E. (2015). Continuum foam: A material point method for shear-dependent flows. *ACM Trans. Graph.*, 34(5):160:1–160:20.

Yue, Y., Smith, B., Chen, P. Y., Chantharayukhonthorn, M., Kamrin, K., and Grinspun, E. (2018). Hybrid grains: Adaptive coupling of discrete and continuum simulations of granular media. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 283:1–283:19.

Zang, G., Idoughi, R., Tao, R., Lubineau, G., Wonka, P., and Heidrich, W. (2019). Warp-and-project tomography for rapidly deforming objects. *ACM Trans. Graph.*

Zarifi, O. and Batty, C. (2017). A positive-definite cut-cell method for strong two-way coupling between fluids and deformable bodies. pages 7:1–7:11.

Zhou, Y., Lun, Z., Kalogerakis, E., and Wang, R. (2013). Implicit integration for particle-based simulation of elasto-plastic solids. *Computer Graphics Forum*, 32(7):215–223.

Zhu, B., Lee, M., Quigley, E., and Fedkiw, R. (2015). Codimensional non-newtonian fluids. *ACM Trans. Graph.*, 34(4):115:1–115:9.

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

Zhu, Y., Sifakis, E., Teran, J., and Brandt, A. (2010). An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.*, 29(2):16:1–16:18.