Viscous Liquid Animation with Spatially Adaptive Grids

by

Yipeng Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

Abstract

Viscous fluid behaviors are among the most complex yet familiar physical phenomena we encounter in everyday life. Much attention and investigation has been paid to the creation of visually realistic results, especially some unique effects such as folding and buckling, in computer graphics. However, simulation of viscous fluids requires more computational resources than its inviscid counterpart, since the viscous solve typically has lower sparsity and more degrees of freedom than the Poisson problem used to compute pressure forces. One interesting feature of viscous fluids is that the most important visual details happen at free surfaces of the fluid, while the interior flow remains relatively smooth. Therefore, a spatially adaptive grid with higher density of cells for fluid surfaces and lower density for the interior can be very useful in reducing computational resources and maintaining high-fidelity imagery at the same time.

The focus of this thesis is to provide a method for simulating a highly viscous liquid on an adaptive quadtree grid, and generating visually plausible results. Aside from reviewing the techniques for viscous fluid simulation in computer graphics, we propose a new finite difference scheme to accurately compute the results at junctions where different levels of the quadtree are adjacent to each other. In addition, we apply the variational approach originally proposed by Batty and Bridson [2008] to this scheme, and generate a symmetric positive definite system on which a preconditioned conjugate gradient solver works very well. Thanks to the variational formulation, our method enforces the boundary condition at viscous free surfaces without the need of extra efforts. Lastly, this thesis presents a new scheme transferring velocities between an adaptive grid and a regular grid, which makes it easy to embed our viscosity solver into any grid-based inviscid fluid solver. We experimentally demonstrate that our method is first order accurate and achieves visual results that are qualitatively consistent with those of dense uniform grids, while reducing the number of degrees of freedom by a factor between 2 and 6, depending on the scenario.

Acknowledgements

First of all, I would like to thank my supervisor Christopher Batty for accepting me as his Master's student, introducing me to this interesting and challenging project, offering guidance and support on a regular basis. Whenever I encounter difficulties in research, he always shows me the way to solve the problem instead of simply giving me a solution. In addition, Prof. Batty spends enormous time and efforts helping me refine this thesis and improve my academic writing skill.

Second, I want to thank my current and formal colleagues at the Computational Motion Group: Jumyung, Ryan, Jade, Omar, Egor, Dustin, Vinicius, and Filipe. Our shared interest in the physics-based animation makes the discussions we have meaningful and insightful. Special thanks go to Ryan Goldade, who gives me plenty of suggestions on not only this research topic, but career planning and work-life balance.

I feel lucky of being a member of the SciCom lab and working with a group of talented and hard-working students. In no particular order: Eddie, Colleen, Parsiad, Ke, Luyu, Bo, Kai, Shan. You together create a friendly and relaxed working environment which does help erase the stress from the daily work.

I am also grateful to all my friends outside the lab, especially Haotian, Jingjie, Luke, Sophie, Jack, and Yuan. You make the life outside of work hours interesting and colorful.

Over the course of the Master's program, I took two short internships with Amazon Web Services and Google. I thank these organizations for giving me the opportunity to visit and work with them. In particular, I would like to acknowledge my hosts at Google, Aaron Xiao and Andrew Grieve, for involving me in impactful projects, reviewing my code and having discussions on technical details. Also, I thank my personal mentor at Google, Owen Glofcheski, for introducing me friends in Mountain View, and suggesting me several different career paths.

Finally, I am indebted to my wife Hao Wang for giving me unconditional support and understanding during the course of the Master's program. It is impossible for me to finish this project without you standing by my side. I want to thank you from the bottom of my heart.

Dedication

To my wife, Hao, and my daughter Catherine.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Fluids are an indispensable part of our life and we interact with them every day: from syrup we put on pancakes in the morning to the coffee we drink in the afternoon. Due to the familiarity we have with fluid behavior, minor inaccuracies in the animation can often be spotted by our eyes. It is no wonder the animation of fluid motion is an important and challenging topic in computer graphics.

Early fluid animation models were driven by visual appearance instead of physical accuracy, and had difficulty in responding to manually added forces and in preserving finely resolved surface details. To conquer these problems, the Navier-Stokes equations, which describe common real-world fluids mathematically, have to be solved. Since aspects of this problem have been studied in the field of computational fluid dynamics, some simulation techniques are borrowed and applied directly to fluid animation in computer graphics. However, these two fields have different objectives: the first focuses on engineering applications and requires high order accurate results of physical quantities such as stresses and forces, while the second investigates the way to create realistic visualizations; numerical accuracy is only important insofar as it supports the goal of apparent realism. There is however one common objective shared by the two fields: minimizing computation resources spent on simulation. This is also a key goal of the present work.

In this thesis, we consider animation of highly viscous (Newtonian) fluids such as honey, molasses, and oils. The viscosity coefficients of such fluids do not need to be constant everywhere but should not depend on the flow velocity. In addition, we will focus on fluids that are not fully surrounded by a solid but rather possess a deforming liquid-air boundary or surface. We will treat this boundary using the so-called free surface assumption, that assumes the fluid moves within a vacuum based on the fact that the air density is negligible compared to the density of common viscous fluids. The alternative would be to additionally simulate the air volume as a second fluid (or phase), using a two-phase fluid simulator. However, this has a much higher computational cost and generates similar results with the single-phase model equipped with the free-surface boundary condition in most scenarios. This further justifies our usage of the free-surface approximation.

For the purposes of animation, viscous forces have little apparent impact on low-viscosity flows such as smoke or water. As such, these can be reasonably approximated by the incompressible Euler equations which ignore both viscous and thermal effects on the flow. However, since we do want to explicitly simulate highly viscous fluids, the incompressible Navier-Stokes equations have to be used. Specifically, the viscosity of the fluid is described by the Stokes' stress constitutive equation which relates the deformation that the fluid is undergoing to the resulting internal forces. The distinguishing characteristic of the viscous fluid is its slow and damped motion due to the presence of the shear stress, which makes the interior of the fluid not extremely interesting to watch. However, unique effects can arise at free-surface boundaries. For example, when a viscous fluid drops down on a solid surface, it will show folding and buckling effects on the fluid surface as illustrated in Figure 1.1. This behavior can be qualitatively explained with the fact that liquid prefers the path where it meets with the least resistance. The viscous fluid below causes much higher force than that applied by the surrounding air (vacuum), so that the fluid always falls in the opposite direction relative to the fluid segment below. Many delicate details are captured by precisely simulating the behavior of the liquid's free surface.

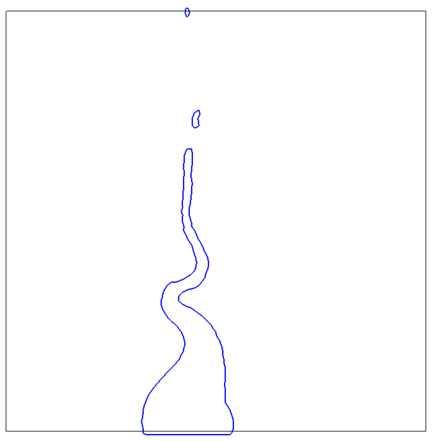
Figure 1.1: A straight stream of a viscous fluid buckles when it falls.

However, as noted by Batty and Bridson [2008], we must pay careful attention to the form of the equations describing viscous forces if we wish to produce the distinguishing effects at free-surface boundaries (we will discuss the details in Chapter 3). The linear system used in the viscosity solve has higher density (lower sparsity) than the Poisson problem typically used to compute pressure forces since it normally requires more variables to update a particular velocity sample. For example, a given matrix row in a standard finite difference discretization of a 2D Poisson problem will involve at most 5 matrix entries;

by contrast, the discretization of [Batty and Bridson, 2008] can involve up to 9 entries per row. Furthermore, because the linear system for solving viscosity is written in terms of the velocity vector (with 2 or 3 components, depending on the spatial dimension being simulated), it is 2 or 3 times larger than the scalar pressure problem. Because of these facts, we observed that the viscosity solve could take as much as 70% of the total simulation time depending on the liquid configuration.

As noted before, we care more about the surface of the viscous fluid where most visually interesting behavior happens. In order to optimize the use of computational resources, we would therefore like to use an adaptive quadtree grid (as illustrated by Figure 1.2) which creates refined cells by evenly dividing a coarse cell into 4 in 2D and places more cells on the fluid surface. In fact, adaptive quadtree (octree) grid strategies for the pressure forces in incompressible fluid are quite common. Popinet [2003] claimed to be the first to simulate incompressible *inviscid* fluid on an octree grid. Losasso et al. [2004] extended that work to free-surface fluids (which is an indispensable step of introducing the octree simulation to computer graphics) and proposed a way to generate a symmetric linear system. Although the accuracy of their results is reduced to first-order, the animation is still visually realistic. Since the viscosity force has a similar mathematical form to the pressure force, we naturally arrived at the idea of simulating viscous fluids on a quadtree grid. Therefore, the main objective of the thesis is efficient generation of visually realistic animation of viscous fluids on a quadtree grid. Ideally, we would also prefer that our method yields a symmetric linear system and offers at least first-order accurate results, as Losasso et al. [2004] did for the pressure solve.

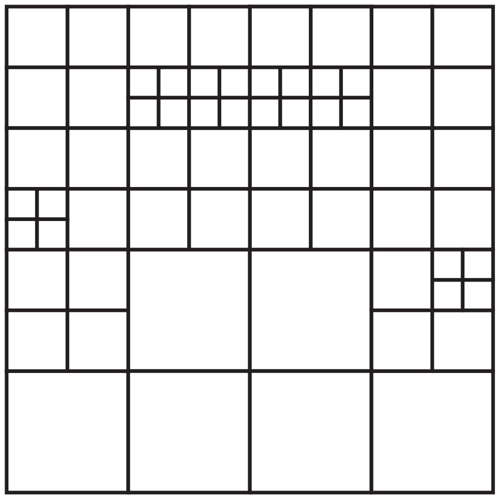

Figure 1.2: A generic graded quadtree grid.

1.1 Contributions

We present a novel method for simulating viscous fluids on a spatially adaptive grid (e.g. quadtree in 2D), which also captures details on the fluid surfaces. The contributions of this thesis are listed as follows:

1. Proposing a new finite difference scheme for simulating viscous fluids on a spatially adaptive grid, and applying a variational formulation to ensure that the method generates a symmetric positive definite (SPD) linear system on which a preconditioned conjugate gradient solver works very well.

2. Proposing a scheme for transferring velocities between an adaptive grid and a regular grid, so it is easy to embed our viscosity solver into any grid-based or hybrid fluid simulator.

3. Evaluating the simulation results in terms of speed and accuracy.

1.2 Outline

The purpose of this chapter has been to briefly introduce the problem we address in this project. The remaining chapters are ordered as follows: Chapter 2 discusses the Navier-Stokes equations and the operator-splitting method, and reviews previous work related to advection, pressure projection and fluid surface representation. The main contribution of this thesis is described in Chapter 3, which first summarizes the related work specific to viscous liquid animation. Then, it introduces the continuous viscosity equation and a variational interpretation, and constructs a linear system by optimizing a discrete energy functional. In addition, it explains how to create a SPD linear system on a quadtree grid. Chapter 4 is dedicated to the detailed implementation of our algorithm and the presentation of results. The thesis is concluded in Chapter 5 in which some potential extensions of the work are also proposed.

Chapter 2

Fluid Animation

In this chapter, we will summarize the governing equations and standard techniques for Eulerian fluid simulation in computer graphics. A more thorough exploration of this subject can be found in the textbook by Bridson [2015]. Since the viscosity solve is the main contribution of this thesis, we will only mention its equation here and leave the detailed explanations to the next few chapters.

2.1 Navier-Stokes equations

Based on the fact that an incompressible fluid material should follow the law of mass conservation and the law of momentum conservation, its motion is given by the following equations:

$$\begin{aligned} \boldsymbol{\nabla} \cdot \vec{u} &= 0 \\ \rho \frac{D\vec{u}}{Dt} &= \boldsymbol{\nabla} \cdot \sigma + \vec{f} \\ \sigma &= -p\delta + \tau \\ \tau &= \mu(\boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla}\vec{u}^T) \end{aligned} \tag{2.1}$$

where $\boldsymbol{\nabla}\cdot$ is the divergence operator, $\vec{u}$ is the fluid velocity, ρ is the fluid density, $\frac{D}{Dt}$ is the material derivative which is equal to $\frac{\partial}{\partial t} + \vec{u} \cdot \boldsymbol{\nabla}$, σ is the Cauchy stress tensor, $\vec{f}$ is the external forces, p is pressure, δ is the identity tensor, τ is the viscous shear stress tensor and μ is the viscosity coefficient. These equations are called the Navier-Stokes equations [Batchelor, 1967]. The first equation enforces incompressibility of the fluid, the second updates the velocity based on fluid motion, internal stresses, and external forces, and the third and fourth determine the state of stress within the fluid.

Although it is not proven whether smooth, physically reasonable analytic solutions exist for the Navier-Stokes equations [Fefferman, 2006], they provide an effective description of the observed physical behaviour of fluids, and researchers have proposed many different ways to solve them numerically. There are two main methods to simulate 3D fluids in computer graphics: Lagrangian particles and Eulerian grids. In a Lagrangian method our reference point moves with the flow, whereas for Eulerian methods the reference frame remains fixed as the fluid flows past. The most widely used particle-based method in computer graphics is smoothed particle hydrodynamics (SPH) [Monaghan, 1992; Müller et al., 2003], which treats the fluid as a particle system, in which each point of the fluid is a particle. It has two main advantages. First, it handles advection with less numerical dissipation than grid-based methods. Since moving a particle does not change its velocity, we can simply simulate it with ordinary differential equation (ODE) solvers. Second, the method preserves some interesting visual details such as splashes and drops without the loss of mass that can occur with other methods. However, it requires a dense sampling of particles to fill in the whole fluid volume. Besides, it has difficulties with interaction forces such as pressure and viscosity leading to instabilities for a large time step size, and some common variants do not strictly guarantee the incompressibility condition [Zhu and Bridson, 2005]. By contrast, Eulerian grid methods offer a straightforward discretization scheme, and relatively easy enforcement of the incompressibility condition. However, they tend to have more difficulties with the advection terms that arise due to the fixed reference frame. Traditionally the most common grid-based method, first order semi-Lagrangian [Stam, 1999], suffers from excessive numerical dissipation due to accumulated interpolation errors. Although better results for the velocity field can be achieved with more sophisticated integration techniques such as high-order Runge-Kutta with higher order interpolation, Eulerian methods for advecting the signed distance field, which is commonly used to represent the fluid surface, have been demonstrated to struggle at maintaining thin details, even with fifth-order accurate schemes [Enright et al., 2002a].

By listing the pros and cons of Eulerian vs. Lagrangian approaches, we can observe that they seem to be complementary, and a better result might be achieved if we mix these two methods together. Foster and Fedkiw [2001] used inertialess particles to enhance the advection of level sets on top of a grid-based simulation. Losasso et al. [2004] extended this particle level set method to a simulation on an adaptive octree. Zhu and Bridson [2005] solved the advection part directly on particles while nevertheless enforcing incompressibility on the grid. This is called the fluid implicit particle (FLIP) method and dramatically reduces numerical diffusion. In this thesis, we use a method similar to [Losasso et al., 2004]. To be specific, we solve the equations of motion on a grid and reconstruct a level set surface at each time step with the help of massless marker particles that passively follow the velocity field. However, as previously mentioned in Section 1.1, it is also easy to integrate our viscosity solver into a FLIP-like hybrid method. Since most components of our simulator are implemented on the grid, we will focus on Eulerian approaches from this point on.

Although it is possible to numerically solve the Navier-Stokes equations as a whole in one step (e.g., Taylor and Hood [1973] and Ghia et al. [1982] accomplished that with a standard finite element method and an implicit multigrid method, respectively), researchers in computer graphics tend to use an operator-splitting method due to its simplicity and efficiency. Equation 2.1 is split up into several components, such that the output of one component becomes the input of another. By arranging these components in a proper sequence, this method produces reasonable results.

Typically, the equations are separated into the pressure part ("pressure projection"), the advection part, the external force part, and the viscosity part. These components are given as follows:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} + \frac{1}{\rho}\boldsymbol{\nabla} p = 0 \\ \text{s.t. } \boldsymbol{\nabla}\cdot\vec{u} = 0 \end{aligned} \tag{2.2}$$

$$\frac{D\vec{u}}{Dt} = 0 \tag{2.3}$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{f} \tag{2.4}$$

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} = \frac{1}{\rho}\boldsymbol{\nabla}\cdot\tau \\ \text{s.t. } \tau = \mu(\boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla}\vec{u}^T) \end{aligned} \tag{2.5}$$

To conserve the fluid volume, we must only advect data using a divergence-free (i.e., incompressible) velocity field; otherwise sources and sinks in the velocity field will cause volume to be created or destroyed. Therefore, the pressure projection (2.2) should be solved just before advection (2.3), since it projects out divergent components of the velocity field. Some authors have advocated for the pressure projection to be performed twice, once before and once after the viscosity solve, so that the input $\vec{u}$ to (2.5) is also divergence-free, which can further reduce error [Losasso et al., 2006b; Batty and Bridson, 2008]. Our experience indicates that in practice this yields little difference in the visual results, so we do not pursue it. We would like to emphasize that the operator-splitting method is simply one way to achieve reasonable results, and is by no means a perfectly accurate one. Actually, recent research has revealed some important visual effects can be erroneously eliminated by this method. For example, Zhang et al. [2015] showed that the advection step transferred some part of the velocity field from divergence-free modes into divergent modes and thus violated the vorticity equation. Larionov et al. [2017] also demonstrated that naively separating the pressure and viscosity solves reduced accuracy in results and produced incorrect free-surface behavior. Nevertheless, we will adopt the standard operator-splitting approach outlined above because it is effective in many relevant scenarios.

2.2 Simulation Grid

Equations 2.2-2.5 involve both time and spatial derivatives. Since the time derivative can be easily discretized with backward Euler, most research focuses on the spatial discretization which normally involves variables in two or three dimensions and thus, partial derivatives. The three most widely used numerical methods to solve partial differential equations (PDE) are the finite element methods (FEM), finite volume methods (FVM) and finite difference methods (FDM). Although all of them have been utilized in computer graphics, such as a viscoelastic fluid simulation with FEM([Bargteil et al., 2007]), a smoke simulation with FVM ([Fedkiw et al., 2001]) and a general fluid simulation with FDM([Stam, 1999]), FDM is inarguably the most popular one due to its simplicity.

Before talking about the fluid simulation routines, we need to discuss the grid used for space discretization. The marker-and-cell (MAC) grid proposed by Harlow and Welch [1965] in computational fluid dynamics (CFD) has become one of the most widely used grid structures in computer graphics. It is also called a staggered grid since the different variables are stored at different locations.

Figure 2.1 shows a typical MAC grid where we use a thick dash to denote a velocity component, a circle to denote a diagonal stress component and a square to denote an off-diagonal stress component. We will use the same symbols in subsequent grids. As an aside, the pressure variable also locates at the grid cell centre since we use the central difference between two neighbouring pressure values to update the velocity in the middle when performing pressure projection.

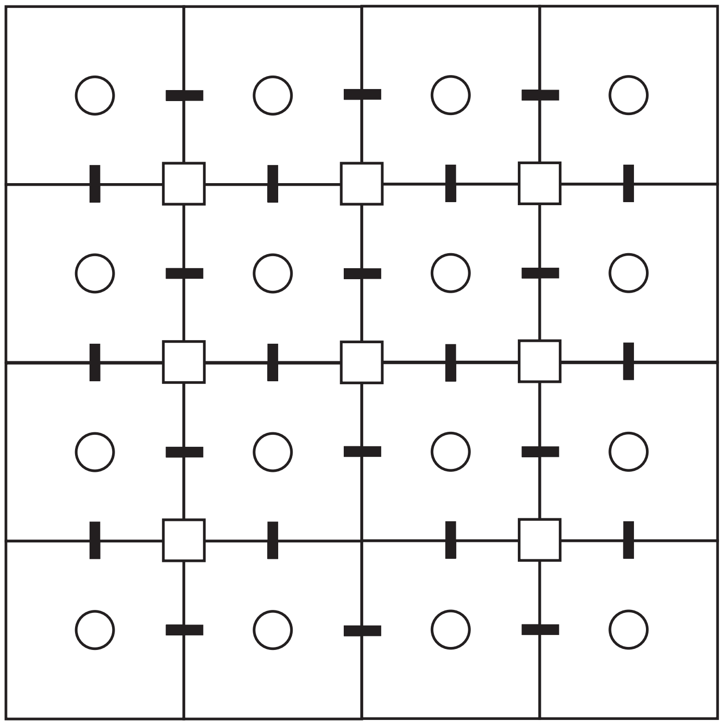

Figure 2.1: The standard staggered grid layout in 2D.

The main reason for using the MAC grid (as opposed to a collocated grid in which all variables are stored at the same positions) is to get rid of a problem caused by the presence of null spaces. For example, the central difference equation calculating the partial derivative with respect to x at $[i, j]$ on a collocated grid is

$$\left(\frac{\partial f}{\partial x}\right)_{i,j} \approx \frac{\mathbf{f}_{i+1,j} - \mathbf{f}_{i-1,j}}{2\Delta x}, \tag{2.6}$$

where $\mathbf{f}_{i+1,j}$ and $\mathbf{f}_{i-1,j}$ are values of f sampled at $[i+1, j]$ and $[i-1, j]$. Assuming $\mathbf{f}_{i-1,j}$ and $\mathbf{f}_{i+1,j}$ are equal, (2.6) will always report the derivative as zero no matter which value $\mathbf{f}_{i,j}$ takes on. We can also view the problem from the perspective of fluid simulation. In the case of 2D, the central difference approximation of the divergence operator $\nabla \cdot \vec{u}$ at $[i, j]$ on a collocated grid is

$$(\nabla \cdot \vec{u})_{i,j} \approx \frac{\tilde{u}_{i+1,j} - \tilde{u}_{i-1,j}}{2\Delta x} + \frac{\tilde{v}_{i,j+1} - \tilde{v}_{i,j-1}}{2\Delta x} = 0, \tag{2.7}$$

where $\tilde{u}$ and $\tilde{v}$ are x-axis and y-axis components of discrete velocities. Suppose both $\tilde{u}_{i,j}$ and $\tilde{v}_{i,j}$ have the form $((-1)^i, (-1)^j)$; then (2.7) does not depend on the velocities at $[i, j]$ and is satisfied, despite the fact that such an oscillatory $\tilde{u}_{i,j}/\tilde{v}_{i,j}$ is clearly divergent. In other words, (2.7) cannot detect any high-frequency divergence in $\tilde{u}_{i,j}$, and the velocity remains highly divergent. As we discussed in Section 2.1, the divergence-free velocity is required for the step of advection to conserve the fluid volume. Therefore, it is not possible to use the central difference method on a collocated grid unless the high-frequency divergent mode is filtered out in advance or explicitly treated in some other fashion. Adopting the staggered grid configuration neatly sidesteps this issue and simplifies the implementation, because the variable placement ensures no null space issues arise with central differencing. Since we use the staggered grid discretization for pressure projection, we also keep this same variable layout for the remaining steps of the simulation, including advection and viscosity. We will further discuss pressure projection in Section 2.5.

2.3 Velocity Advection

Stam first introduced the semi-Lagrangian scheme to computer graphics [Stam, 1999], which has been popular in atmospheric science for a long time [Staniforth and Côté, 1991]. Although the word Lagrangian appears in its name, the method is actually grid-based (Eulerian).

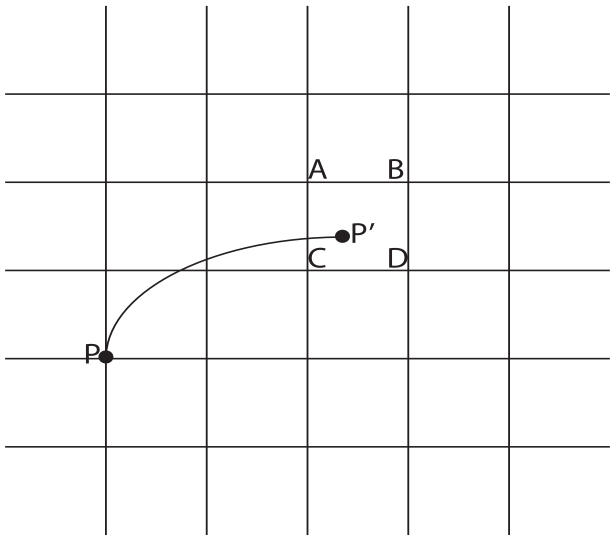

Figure 2.2: A semi-Lagrangian update of data at position P is based on tracing the velocity field backwards to point P' and interpolating from the surrounding nodes.

As we discussed before, the reason why particle-based advection can be solved with an ODE solver is that in the absence of source terms the change rate of the particle value is zero. We can apply the same principle to grid-based advection. In order to figure out the new value at the grid point P, we assume there will be a particle arriving at P at the time $t^n + \Delta t$. Since the particle value doesn't change with time according to (2.3), we just need to find the particle's location P' at the time t^n and assign its value to the location P. If the point P' does not fall on any grid point (as illustrated in Figure 2.2), we use simple bilinear interpolation, separately for each component of the velocity. Since a hypothetical Lagrangian particle is used to figure out how to advect on an Eulerian grid, this is called a semi-Lagrangian method.

The remaining problem is to determine how we can find the location P' by following the velocity field. We know that the particle's motion follows a simple ODE:

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{u}(\vec{x}).$$

By discretizing the derivative $\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}$ with forward Euler and integrating backwards in time, we get

$$\vec{x}_{P'} = \vec{x}_P - \Delta t \vec{u}(\vec{x}). \tag{2.8}$$

The result is sometimes adequate, but we can only trace along a straight trajectory with (2.8). For particles around swirls or other rotational flow elements, we have to adopt a higher order integration technique such as second- or third-order Runge-Kutta methods to obtain better results.

Another issue we need to pay attention to is the size of Δt. Most numerical methods have the stability concern of whether the numerical error will aggregate and finally blow up. However, this is not a problem for the semi-Lagrangian method because all new values are simply (bi)linearly interpolated results of old values, which means it's impossible to create any value larger or smaller than existing ones. This conclusion holds no matter how large the time step is and that is why the method is unconditionally stable. In practice, an aggressive time step brings extra numerical dissipation not to mention reduced accuracy. Foster and Fedkiw [2001] suggested a good rule of thumb would be to limit the time step such that we don't allow data to traverse more than 5 grid cells in one time step, where the value 5 in this case is referred to as the CFL (Courant-Friedrichs-Lewy) number. In other words, the time step must satisfy the following condition:

$$\Delta t \leq \frac{5\Delta x}{|\vec{u}|_{max}}. \tag{2.9}$$

Some researchers tend to use even smaller time steps, such as a CFL number of 1, to get more accurate results.

2.4 Fluid Surface Representation

Before discussing the pressure and viscosity solves, we need a technique to tell which cells are on the fluid boundaries (partially covered by a fluid), which cells are on the interior, and which are in the (empty) air region. One of the most commonly used methods in computer graphics is the level set method [Fedkiw and Osher, 2003]. The formal definition of the level set in $\mathbb{R}^n$ is given below:

$$\Gamma = \{\vec{x} | \phi(\vec{x}) = c\}, \tag{2.10}$$

where $\vec{x}$ is an n-dimensional position vector, ϕ is the level set function, c is a constant, and Γ represents the set of points in $\mathbb{R}^n$ where the level set function $\phi(x_1, \ldots, x_n)$ takes on the given value c.

To define a fluid surface, we just need to choose a threshold value of $\phi(\vec{x})$ for all surface points. Suppose we select zero as the threshold for the 2D simulation; then,

- $\phi(\vec{x}) = 0$ if the point $\vec{x}$ is on the surface,
- $\phi(\vec{x}) > 0$ if the point $\vec{x}$ is outside the fluid (solid or air), and
- $\phi(\vec{x}) < 0$ if the point $\vec{x}$ is inside the fluid.

Note that (1) it is not necessary to set the fluid surface threshold to zero, although this is normally the way it is set up in fluid simulation [Bridson, 2015]; (2) the negative and positive regions can be reversed ($\phi(\vec{x}) > 0$ for the inside fluid points) since this choice is arbitrary. In this project, we use the signed distance function to describe the implicit surface because the form of the function is very simple and we can later use the signed distance at each grid point to aid us in constructing an adaptive grid. Suppose S is the set of all points $\vec{s}$ on the fluid surface. Then the signed distance function is defined as

$$\phi(\vec{x}) = \begin{cases} \min\limits_{\vec{s} \in S} \|\vec{x} - \vec{s}\| : \vec{x} \text{ is outside;} \\ -\min\limits_{\vec{s} \in S} \|\vec{x} - \vec{s}\| : \vec{x} \text{ is inside.} \end{cases} \tag{2.11}$$

We are discretizing the level set onto a grid, and thus the surface is the set of all points whose interpolated value is zero.

Now a simple way to make use of a level set in simulation is to define $\phi(\vec{x})$ at each cell centre at t_0 as an initial state, and advect the level set with the semi-Lagrangian method described in Section 2.3. However, this method can cause some issues even with a high-order advection scheme on a grid [Enright et al., 2002b]. As we know from our daily experience, it is normal for a fluid to develop many thin structures such as droplets and splashes. But a level set on the grid cannot correctly address any structure thinner than two cells [Bridson, 2015], and these details will soon disappear as the simulation proceeds. An intuitive way to solve the problem is to use Lagrangian advection which introduces much less numerical diffusion. Here we introduce two widely adopted approaches in computer graphics.

The first one is called the marker particle method, used originally in [Harlow and Welch, 1965] which also proposed the MAC grid discussed in Section 2.2. In this method, we first generate water particles to fill the volume of water in a random jittered pattern. Specifically, we ensure there are always two particles within each cell but their locations are sampled from a uniform distribution. This is recommended when emitting marker particles because sampling on a regular lattice may lead to anisotropically biased stripe-like particle distributions in certain situations [Bridson, 2015]. Then, we move the particles according to the grid velocity field and use them to determine which grid cells should be included in the simulation in the next time-step. A problem with this approach is the question of how to render a smooth surface from a group of particles. Blinn [1982] introduced the "blobby" method where a user can adjust the "blobbiness" of the object. The level set function can be defined from a collection of particles using

$$\phi(\vec{x}) = \sum_i k\left(\frac{\|\vec{x} - \vec{x}_i\|}{h}\right), \tag{2.12}$$

where $\vec{x}_i$ is the location of particle i, k is a suitable smooth kernel function and h is the spatial extent (i.e., essentially its radius of influence) of each particle. An intuitive choice for k is a Gaussian distribution. A simpler alternative would be a spline like

$$k(s) = \begin{cases} (1-s^2)^3 : s < 1; \\ 0 : s \geq 1. \end{cases} \tag{2.13}$$

Here $s = \|\vec{x} - \vec{x}_i\|/h$, and the extent h is normally several times the average inner particle spacing r. The threshold value of the level set is set to $k(r/h)$ instead of zero in their method, and the surface looks like many spheres smoothly connecting together. An issue with this approach is the surface looks, as its name suggests, too blobby. It is possible to smooth out the artifacts by increasing the size of h (spheres around particles will look larger), but this may also cancel out some small scale features. Selecting an appropriate value of h turns out to be a key issue in making the “blobby” method work well for liquids. Zhu and Bridson [2005] improved this method by using a distance-based surface model

$$\phi(\vec{x}) = \left\|\vec{x} - \bar{X}\right\| - \bar{r}, \tag{2.14}$$

where $\bar{X}$ and $\bar{r}$ are a weighted average of nearby particle locations and radii, respectively:

$$\begin{aligned} \bar{X} &= \frac{\sum_i k\left(\frac{\|\vec{x}-\vec{x}_i\|}{h}\right)\vec{x}_i}{\sum_i k\left(\frac{\|\vec{x}-\vec{x}_i\|}{h}\right)} \\ \bar{r} &= \frac{\sum_i k\left(\frac{\|\vec{x}-\vec{x}_i\|}{h}\right) r_i}{\sum_i k\left(\frac{\|\vec{x}-\vec{x}_i\|}{h}\right)}. \end{aligned} \tag{2.15}$$

We can simply set all the particles’ radii to the average inter-particle spacing r, and $\bar{r}$ will also be equal to r. A better result on flat surfaces can be achieved by using the actual particle-to-surface distance as the particle’s radius. Finally, the fluid surface is defined as a set of points $\vec{x}$ where $\phi(\vec{x})$ given in (2.14) is equal to zero. Although this method removes most blobby artifacts, it could potentially introduce some small-scale artifacts at concave regions where the $\bar{x}$ may sit outside the surface and an extra smoothing on the level set may be required.

Another common method of appropriately advecting the level set is called the particle level set [Enright et al., 2002b], which advects a band of massless marker particles around the fluid surface along with the implicit function on the grid, and uses the local level set created from the particles to correct errors in the surface represented by the implicit function. An advantage of this method is that we just need to modify the existing level set instead of creating a new one from particles so it doesn’t have the “blobby” issues with the marker particle method.

In this project, we use the marker particle method similar to the one in [Zhu and Bridson, 2005] but with a different level set function. Suppose A is a set of particles near the grid point $\vec{x}$; the level set function would be

$$\phi(\vec{x}) = \min_{\vec{x}_i \in A} \|\vec{x} - \vec{x}_i\| - r, \tag{2.16}$$

which is just the distance field for a union-of-spheres approximation of the surface and is selected for its simplicity. Both (2.14) and (2.16) give the exact signed distance for the region outside the fluid but produce no meaningful results for the interior. Therefore, we have to apply the fast marching method [Tsai, 2002] to propagate the signed distance from the surface to the interior.

2.5 Pressure Projection

In this section, we will briefly discuss an essential part of a fluid simulation, the pressure projection step making the fluid incompressible and satisfying boundary conditions. By applying time discretization to (2.2), we have

$$\vec{u} = \vec{u}^{old} - \frac{\Delta t}{\rho}\boldsymbol{\nabla} p \tag{2.17}$$

which subtracts the pressure gradient from an intermediate velocity and generates a divergence free result such that

$$\boldsymbol{\nabla} \cdot \vec{u} = 0. \tag{2.18}$$

At the same time, the pressure should satisfy the free surface boundary condition

$$p = 0 \tag{2.19}$$

at the liquid-air surface and the velocity should satisfy the solid wall boundary condition

$$\vec{u} \cdot \vec{n} = \vec{u}_{solid} \cdot \vec{n} \tag{2.20}$$

along any solid boundaries. The free surface boundary condition ensures there is no pressure force pushing from the air onto the liquid (consistent with the assumption of negligible density air), and the solid wall boundary condition prevents fluid from flowing into or out of the walls. At typical human scales in the physical world, we can readily observe that fluid does not quite stick to walls as the above simple condition would dictate. Instead, liquid easily separates from walls, though it may leave behind a thin film or small droplets. Some researchers intentionally modified (2.20) to simulate this wall-separation phenomenon. For example, Batty et al. [2007] proposed the condition:

$$0 \le p \perp (\vec{u} - \vec{u}_{solid}) \cdot \vec{n} \ge 0. \tag{2.21}$$

It states that either both $p > 0$ and $(\vec{u} - \vec{u}_{solid}) \cdot \vec{n} = 0$ are true or both $p = 0$ and $(\vec{u} - \vec{u}_{solid}) \cdot \vec{n} \ge 0$ are true which ensures that pressure prevents fluid from entering walls, while allowing it to freely separate. We just use (2.19) in this project since it is easy to implement and the wall-separation phenomenon does not typically happen for highly viscous fluids.

As we discussed before, spatial discretization (e.g., via finite differences) is often fairly straightforward for Eulerian grid methods, as compared to particle methods. By substituting (2.17) in (2.18), we reach the equation to determine the pressure

$$\frac{\Delta t}{\rho}\nabla^2 p = -\boldsymbol{\nabla} \cdot \vec{u}^{old}. \tag{2.22}$$

The central finite difference approximation of (2.22) for one cell on a 2D MAC grid is given as follows:

$$\begin{aligned}\frac{\Delta t}{\rho}\left(\frac{4\tilde{p}_{i,j}-\tilde{p}_{i+1,j}-\tilde{p}_{i-1,j}-\tilde{p}_{i,j+1}-\tilde{p}_{i,j-1}}{\Delta x^2}\right) &= \\ -\left(\frac{\tilde{u}^{old}_{i+1/2,j}-\tilde{u}^{old}_{i-1/2,j}+\tilde{v}^{old}_{i,j+1/2}-\tilde{v}^{old}_{i,j-1/2}}{\Delta x}\right)\end{aligned} \tag{2.23}$$

where $\tilde{p}$ represents the discrete pressure. We can build a linear system based on (2.23) and solve it with a standard linear system solver such as preconditioned conjugate gradient. Then, we use (2.17) to update the velocities:

$$\begin{aligned}\tilde{u}_{i+1/2,j} &= \tilde{u}^{old}_{i+1/2,j}-\frac{\Delta t}{\rho}\frac{\tilde{p}_{i+1,j}-\tilde{p}_{i,j}}{\Delta x}, \\ \tilde{v}_{i,j+1/2} &= \tilde{v}^{old}_{i,j+1/2}-\frac{\Delta t}{\rho}\frac{\tilde{p}_{i,j+1}-\tilde{p}_{i,j}}{\Delta x}.\end{aligned} \tag{2.24}$$

To address the free surface boundary condition $p = 0$, we use the standard ghost fluid method proposed by Gibou et al. [2002]. Suppose the cell $[i+1, j]$ is in the fluid $(\phi(\vec{x}_{i+1,j}) < 0)$ and the cell $[i, j]$ is in the air $(\phi(\vec{x}_{i,j}) > 0)$. Instead of setting the pressure at $[i, j]$ to 0, we use linear interpolation to find the surface at $[i + \theta, j]$ where the θ is defined as

$$\theta = \frac{\phi(\vec{x}_{i,j})}{\phi(\vec{x}_{i,j})-\phi(\vec{x}_{i+1,j})}, \tag{2.25}$$

and set the pressure at this location to 0:

$$\tilde{p}_{i+\theta,j} = (1-\theta)\tilde{p}_{i,j}+\theta\tilde{p}_{i+1,j} = 0. \tag{2.26}$$

Based on (2.25) and (2.26), we can easily express $\tilde{p}_{i+1,j}$ with $\tilde{p}_{i,j}$

$$\tilde{p}_{i+1,j} = -\frac{1-\theta}{\theta}\tilde{p}_{i,j}. \tag{2.27}$$

Since the cell $[i+1, j]$ can be imagined as a virtual fluid cell with a pressure value designed to enforce the desired boundary condition, the method is called the ghost fluid method.

By substituting $\tilde{p}_{i+1,j}$ with (2.27) in (2.24), we have a new equation to update $\tilde{u}$ samples that are close to the free surface:

$$\tilde{u}_{i+1/2,j} = \tilde{u}^{old}_{i+1/2,j}-\frac{\Delta t}{\rho}\frac{\phi(\vec{x}_{i+1,j})-\phi(\vec{x}_{i,j})}{\phi(\vec{x}_{i,j})}\frac{\tilde{p}_{i,j}}{\Delta x}. \tag{2.28}$$

We can update $\tilde{v}$ in the same way. As an aside, we will later apply a similar method to the solid wall boundary condition for viscosity, i.e., $\vec{u} = \vec{u}_{solid}$.

The solid wall boundary condition (2.20) is more difficult to enforce since it actually specifies the normal derivative of pressure, $\frac{\partial p}{\partial \vec{n}} = \nabla p \cdot \vec{n}$, rather than the value of pressure. Foster and Metaxas [1996] first explored this topic in computer graphics by voxelizing solid objects onto the grid. However, it only worked well when all object boundaries are perfectly aligned with the grid, and otherwise would introduce stair-step artifacts. Foster and Fedkiw [2001]; Houston et al. [2003]; Rasmussen et al. [2004] attempted to mitigate this issue with different methods, but were only successful in simulating some scenarios because they still used the voxelized pressure solve. Roble et al. [2005] first derived a high quality solid interaction on a standard MAC grid by adding face area weights to the finite difference stencil, which actually turns it into a finite volume approximation. Batty et al. [2007] proposed a variational interpretation of pressure which handled the solid wall boundary condition implicitly. When the solid objects are static, their approach yields the same discretization as [Roble et al., 2005] except for using the mass fraction of the fluid as the weights. Ng et al. [2009] pointed out that the face area weights should be preferred since this yielded second-order accuracy in pressure whereas the cell volume weights (mass fraction) yielded only first-order accuracy. Therefore, we use the same finite volume method as presented in [Roble et al., 2005]. Roble et al. [2005] didn't treat moving objects, so really our implementation is more like [Ng et al., 2009].

We just need to add the face area weights to the original finite difference discretization (2.23) to produce the finite volume stencil which looks like

$$\begin{aligned}
&\frac{\Delta t}{\rho}\Bigg(\frac{F_{i-1/2,j}+F_{i+1/2,j}+F_{i,j-1/2}+F_{i,j+1/2}}{\Delta x}\tilde{p}_{i,j} \\
&\quad -\frac{F_{i-1/2,j}}{\Delta x}\tilde{p}_{i-1,j}-\frac{F_{i+1/2,j}}{\Delta x}\tilde{p}_{i+1,j}-\frac{F_{i,j-1/2}}{\Delta x}\tilde{p}_{i,j-1}-\frac{F_{i,j+1/2}}{\Delta x}\tilde{p}_{i,j+1}\Bigg) \\
&\quad = F_{i-1/2,j}\tilde{u}^{old}_{i-1/2,j}-F_{i+1/2,j}\tilde{u}^{old}_{i+1/2,j}+F_{i,j-1/2}\tilde{v}^{old}_{i,j-1/2}-F_{i,j+1/2}\tilde{v}^{old}_{i,j+1/2} \\
&\quad +(1-F_{i-1/2,j})\tilde{u}^{old}_{i-1/2,j}|_s-(1-F_{i+1/2,j})\tilde{u}^{old}_{i+1/2,j}|_s \\
&\quad +(1-F_{i,j-1/2})\tilde{v}^{old}_{i,j-1/2}|_s-(1-F_{i,j+1/2})\tilde{v}^{old}_{i,j+1/2}|_s
\end{aligned} \tag{2.29}$$

where $\tilde{u}|_s$ and $\tilde{v}|_s$ are the velocities of solid objects, and F is the fluid face area fraction in the range $[0, 1]$ so $F\Delta x$ is equal to the boundary length of a cell covered by the fluid in 2D. We can simply use linear interpolation to find the F in 2D in the same way we retrieved the location of the free surface in (2.25). Bridson [2015] suggested a similar method to compute F in 3D in which a square face was divided into 4 sub-triangles and the fluid area in each triangle was calculated with linear interpolation.

Up to now, we have briefly discussed the Navier-Stokes equations, reviewed several common techniques for the fluid advection, the pressure projection, and the free-surface representation in computer graphics, and emphasized on the methods used in the project. Later in Chapter 4, we will present how to combine all these components together along with the innovative viscosity solve introduced by Chapter 3.

Chapter 3

Viscous Fluid

3.1 Related Work

We have already discussed most components of a fluid simulation such as advection, pressure projection and surface representation, and in this chapter we will focus on the remaining routine, viscosity integration. Unlike the other steps in our solver which have used a regular grid, we will consider solving viscosity on an adaptive quadtree grid, which is the central contribution of this work. We will first review previous research on viscous liquid animation and fluid simulation on an adaptive grid. Then, we will discuss the continuous Laplacian and full forms of viscosity and introduce a variational interpretation of the full form of viscosity. We will emphasize the discretization of the variational form on a staggered grid, and illustrate the required control volumes on a graded quadtree grid in different scenarios. Finally, we will show how we can incorporate our proposed adaptive grid viscosity routine into a standard regular grid fluid simulator.

3.1.1 Viscous Liquid Animation

Although researchers in CFD have used the Navier-Stokes equations to investigate viscous fluid for a long time [Harlow and Welch, 1965], it was not introduced to the field of computer graphics until the work of Foster and Metaxas [1996]. Before that, all approaches in computer graphics were driven more by ad hoc models focused on visual appearance. For example, Miller and Pearce [1989] proposed a particle-based system where viscous springs between particles were modelled to achieve dynamic motions of viscous fluid, and Platt and

Fleischer [1989] used a molecular dynamics approach to simulate melting deformable solids. A main issue with Foster and Metaxas' approach was that each time step was restricted by the local fluid velocity due to the use of explicit integration. This is inconvenient in an animation context, since for simplicity and efficiency we would prefer to be able to choose the time step without fearing loss of stability. Stam proposed an implicit viscosity solve in [Stam, 1999] which enabled larger time steps and which was much more stable than previous work. However, it assumed the viscosity coefficient was constant and thereby decoupled equation (2.5) into three (independent) heat equations, one for each component of velocity. Moreover, it did not consider the role of liquid surfaces, so only gases or completely enclosed liquids could be handled. Foster and Fedkiw [2001] introduced the use of the level set method to Stam's approach, which enabled accurate tracking of the liquid surface. Carlson et al. [2002] further expanded this model by considering heat diffusion, and produced impressive animations of some viscous materials such as melting wax and sand drip castles. While this decoupled (Laplacian) form for viscosity was very commonly used, it unfortunately simplified the proper variable viscosity term and the free surface boundary condition, and thus, introduced nonphysical damping of ballistic motion. Fält and Roble [2003] pointed out that a Neumann boundary condition $\frac{\partial \vec{u}}{\partial \vec{n}} = 0$ produced correct translational motion. Instead of dropping the viscosity terms that should couple the components of velocity together (as done by Carlson et al. [2002] and earlier authors), Rasmussen et al. [2004] eliminated the coupling in the linear system by first integrating the dimensionally coupled components explicitly, and then implicitly integrated the remaining decoupled, symmetric components. In regions of the flow with constant viscosity coefficient, the explicit integration terms cancel, such that the model reduces to three separate linear systems as before. This is an example of an implicit-explicit (IMEX) integration scheme. Hong and Kim [2005] successfully simulated two-phase fluids with discontinuous jumps in viscosity coefficient by enforcing the no-slip condition on the interface between two viscous materials with the ghost fluid method on an octree grid. In contrast to our proposed method, however, this scheme requires velocities to be averaged to cell centres to compute viscous forces, and it does not handle viscosity coefficients that vary smoothly in space. Losasso et al. [2006b] extended this work to multiple immiscible viscous fluids by simulating region-wise constant viscosity with the IMEX scheme and using a separate particle level set method for each region, although on a regular grid. The first fully implicit viscosity solver without dropping the coupling term was introduced to computer graphics by Batty and Bridson [2008]. They created a variational formulation of the viscosity, which treated the complex boundary conditions naturally via simple control volumes. Larionov et al. [2017] applied the same idea to Stokes flows to produce more realistic coiling effects.

3.1.2 Fluid Simulation on Adaptive Grids

Although research on quad/octree-based discretization for compressible fluids was conducted early in CFD [Coirier, 1994; Khokhlov, 1998], the extension to incompressible fluids was first considered by Popinet [2003]. He modified the previously developed adaptive mesh refinement (AMR) schemes [Sussman et al., 1999] for Poisson problems and generated second-order accurate results in space. Losasso et al. [2004] added the support for free surfaces to this approach, and simplified it by calculating the pressure projection with unaligned pressure samples at T-junctions (i.e., locations where changes in grid resolution occur). This sacrificed accuracy in exchange for a simpler SPD system which could be easily solved with the conjugate gradient method. Their method was widely included in applications such as bubbles [Hong and Kim, 2005], solid-fluid coupling [Guendelman et al., 2005], and lightning [Kim and Lin, 2007]. In particular, Hong and Kim [2005] simulated the viscosity term on the octree data structure which was the first attempt in computer graphics. However, they did not discuss the details of the octree viscosity discretization explicitly in the paper. Since the decoupled model was adopted, the numerical stencil for the viscosity was exactly the same as that for the pressure and they used the same discretization as the pressure projection. Losasso et al. [2006a] later improved the discretization of the pressure projection to second-order accuracy by carefully modifying the pressure gradient stencil used at T-junctions while preserving positive-definiteness. Ferstl et al. [2014] and Nielsen and Bridson [2016] used FEM on octree and tile-tree grids, respectively, with extra attention paid to the boundary conditions, because the previously developed finite volume octree scheme had difficulty in providing matrix symmetry, second-order accuracy and support for non-axis aligned boundary condition at the same time. Batty [2017] applied a novel quadratic interpolation to get the value of the missing degree-of-freedom at the T-junction and generated second-order accurate solutions and gradients for Poisson problems, at the cost of losing matrix symmetry. Aanjaneya et al. [2017] utilized power diagrams on parallelism-optimized data structures (the Sparse Paged Grid or ”SPGrid”) [Setaluri et al., 2014] and a multigrid-preconditioner for conjugate gradient to efficiently perform a second-order accurate pressure projection.

Despite the fact that the simulation of viscosity on an adaptive quadtree/octree grid has not been thoroughly explored in computer graphics (to the best of the author’s knowledge, [Hong and Kim, 2005] is the only paper mentioning this topic), researchers from the computational physics community have proposed several different methods. For example, Guittet et al. [2015] explicitly computed a Voronoi diagram from all the u-faces and solved the viscosity on it to get a second-order accurate result. Since the decoupled model was used, they were able to update the v-face results separately in the same way. Gerya et al.

[2013] created some extra "slave" nodes at the T-junction following two principles: (1) the volume flux across resolution levels should be conserved; (2) the stress-based interpolation of velocity gradients should be consistent. With the help of these virtual nodes, they could finally achieve a second-order accurate result for the full viscosity model (i.e. incorporating dimensionally-coupled terms and supporting spatially varying viscosity). However, neither work treated the free surface boundary condition. In addition, variable viscosity could not be addressed by [Guittet et al., 2015] while the system built by [Gerya et al., 2013] was not SPD.

Since more attention is normally paid to realistic visual effects and simulation speed than physical accuracy in computer graphics, we will not concern ourselves with achieving second-order accuracy. However, we would like to have a SPD system which offers:

- Support for spatially varying viscosity;
- Correct free surface behavior;
- Support for spatial adaptivity using a quadtree grid structure;
- First order accuracy across jumps in grid resolution (T-junctions).

3.2 Continuous Viscous Fluid

Any motion of Newtonian incompressible fluids can be mathematically described by equations (2.1). By introducing another physical quantity called the deformation rate tensor,

$$\dot{\varepsilon} = \frac{(\nabla \vec{u} + \nabla \vec{u}^T)}{2}, \tag{3.1}$$

we can rewrite the shear stress tensor as:

$$\tau = 2\mu\dot{\varepsilon}. \tag{3.2}$$

As its name suggests, $\dot{\varepsilon}$ measures how fast the total deformation of the continuum is changing [Bridson, 2015]. The overdot notation implies time derivative, since the deformation rate (or strain rate) is essentially the time derivative of the strain tensor ε.

If μ in equation (3.2) is not a function of the magnitude of the strain rate tensor (however, it can be a function of time or space), τ is proportional to $\frac{1}{2}(\nabla \vec{u} + \nabla \vec{u}^T)$ and this linear relationship is called Newtonian. Fluids with this characteristic are called Newtonian fluids, and all others are categorized as non-Newtonian fluids. This thesis only discusses Newtonian fluids.

By separating the pressure and viscosity solves through operator splitting, as discussed in Section 2.1, the second equation of (2.1) is simplified to the following form:

$$\frac{\partial \vec{u}}{\partial t} = \frac{1}{\rho}\boldsymbol{\nabla} \cdot \big(\mu(\boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla}\vec{u}^T)\big). \tag{3.3}$$

This describes the rate of change of the fluid velocity due to viscous forces alone, as measured by the divergence of the deformation rate. In essence, the faster the fluid is locally deforming, the more the viscous forces act to counter this deformation and damp the flow.

3.2.1 Constant Viscosity

If μ is constant, then the right side of (3.3) can be expanded as $\frac{\mu}{\rho}(\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\vec{u}^T)$. The second term in a 3D Cartesian coordinate system is

$$\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\vec{u}^T = \begin{pmatrix} \frac{\partial}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial}{\partial z}\frac{\partial w}{\partial x} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial}{\partial z}\frac{\partial w}{\partial y} \\ \frac{\partial}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial}{\partial y}\frac{\partial v}{\partial z} + \frac{\partial}{\partial z}\frac{\partial w}{\partial z} \end{pmatrix},$$

where u, v and w are continuous velocity components in the x-, y- and z- directions, respectively. By changing the order of partial derivatives, it becomes

$$\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\vec{u}^T = \begin{pmatrix} \frac{\partial}{\partial x}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \\ \frac{\partial}{\partial y}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \\ \frac{\partial}{\partial z}(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \end{pmatrix} = \boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \vec{u}). \tag{3.4}$$

Since we only investigate incompressible fluid, i.e., $\boldsymbol{\nabla} \cdot \vec{u} = 0$, the second term of the right side of (3.3) vanishes and it turns into

$$\frac{\partial \vec{u}}{\partial t} = \frac{\mu}{\rho}\boldsymbol{\nabla} \cdot (\boldsymbol{\nabla}\vec{u}) = \frac{\mu}{\rho}\nabla^2\vec{u}. \tag{3.5}$$

We will refer to (3.5) as the Laplacian form of viscosity, to distinguish it from the full form given by (3.3). The Laplacian form is straightforward to solve, since it can be discretized in a manner similar to the pressure projection. It is effectively a simple heat equation applied to each component of velocity independently and that is why it is also referred to as the decoupled form in our discussion of related work [Stam, 1999; Carlson et al., 2002]. However, it is only valid when both the viscosity is constant and the velocity field is incompressible [Bridson, 2015].

Although we have proved the Laplacian formulation is adequate to depict constant viscosity fluids, Limache et al. [2007] observe that imposing the natural boundary condition of the Laplacian form on free surfaces causes a violation of the principle of objectivity in continuum mechanics. We will cover more about this topic in the next section.

3.2.2 Boundary Conditions

At solid surfaces, the fluid velocities should be set equal to the solid velocities explicitly according to the no-slip condition. Since this step is straightforward and has been successfully implemented in previous work [Rasmussen et al., 2004; Batty and Bridson, 2008; Larionov et al., 2017], we mainly focus on the free surface boundary condition here.

The correct free surface boundary condition ensures there is no traction $\vec{t}$ applied to the plane of the surface if no surface tension is present [Batty and Bridson, 2008]. Therefore, based on the definition of stress,

$$\vec{t} = \sigma\vec{n} = 0 \text{ on } \Gamma_f$$

where $\vec{n}$ is the outward unit normal of the free surface and σ is the Cauchy stress tensor. By replacing σ with $-p\delta + \tau$ as in (2.1), we have

$$(-p\delta + \tau)\vec{n} = 0, \tag{3.6}$$

where p is the pressure on the free surface, δ is the identity tensor and τ is the shear stress tensor.

The first correct implementation of the condition (3.6) in the context of regular grid methods for computer graphics had not happened until a recent paper [Larionov et al., 2017] was published. They replace the standard viscosity and pressure steps with a novel Stokes step and accurately calculate the free-surface boundary condition with the help of a variational principle. Before that, most research on Eulerian fluid animation was based on decoupled pressure and viscosity solves and the assumption that the free surface pressure is zero [Batty and Bridson, 2008; Batty and Houston, 2011], which reduces (3.6) to the following form:

$$\tau\vec{n} = 0 \text{ on } \Gamma_f. \tag{3.7}$$

While this cannot precisely recover the coiling effects shown by Larionov et al. [2017], it does capture visually plausible viscous flow and buckling behavior in most scenarios of interest to computer animation. It is also the most common approach in industrial fluid animation tools, such as Houdini [SideFX, 2017]. We therefore adopt this boundary condition in this thesis.

3.2.3 Variational Interpretation

The problem we are trying to solve is the PDE (3.3) with the boundary conditions given by (3.7). By discretizing $\vec{u}$ in time with backward Euler, we end up with the following PDE with only space derivatives remaining:

$$\vec{u} = \vec{u}^{\text{old}} + \frac{\Delta t}{\rho}\boldsymbol{\nabla}\cdot\big(\mu(\boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla}\vec{u}^T)\big). \tag{3.8}$$

Batchelor [1967] discusses the minimum dissipation theorem that a correct steady state Stokes flow corresponds to the flow field with the same boundary conditions having the minimum rate of viscous energy dissipation. Batty and Bridson [2008] use this idea to motivate their formulation. Recalling that $\dot{\varepsilon}$ is the deformation rate tensor defined in (3.1), we can define the rate of viscous dissipation Φ as

$$\Phi = 2\mu\dot{\varepsilon} : \dot{\varepsilon} = 2\mu\|\dot{\varepsilon}\|_F^2, \tag{3.9}$$

where the : operator represents a tensor double dot product and $\|\cdot\|_F$ refers to the Frobenius norm of a matrix that is simply the square root of the sum of squares of each element. Batty and Bridson [2008] showed that an equivalent optimization formulation for (3.8) is given as a balance between minimizing this rate of dissipation and minimizing the change in the velocity field. Combining the two conditions together, we reach the following functional:

$$E[\vec{u}] = \iiint_\Omega \rho\left\|\vec{u} - \vec{u}^{old}\right\|^2 + 2\Delta t \iiint_\Omega \mu \left\|\frac{\boldsymbol{\nabla}\vec{u} + \boldsymbol{\nabla}\vec{u}^T}{2}\right\|_F^2. \tag{3.10}$$

Since (3.10) is convex and quadratic in terms of the velocity u, finding the minimizer (of its discretized form) will naturally require solving only a single, symmetric positive definite linear system.

Batty and Bridson [2008] provide an analytic proof that the velocity field minimizing $E[\vec{u}]$ is exactly the solution to equations (3.3 and 3.7). We will therefore proceed to discretize the functional 3.10, which will allow us to easily handle both the free surface conditions and the T-junctions introduced by the quadtree structure.

3.3 Discretization

3.3.1 Discretization of the Stress Tensor

In this section, we will discuss the discretization of the shear stress tensors τ of the Laplacian form (3.5) for completeness and the full form (3.3) on a uniform-resolution MAC grid. In two dimensions, τ is a 2×2 tensor whose components are τ_{xx}, τ_{yy}, τ_{xy}, and τ_{yx} $\left(\tau = \begin{bmatrix} \tau_{xx} & \tau_{xy} \\ \tau_{yx} & \tau_{yy} \end{bmatrix}\right)$. The diagonal stress components τ_{xx} and τ_{yy} are sampled at cell centres, and the off-diagonal ones τ_{xy} and τ_{yx} are sampled at cell corners as illustrated in Figure 3.1.

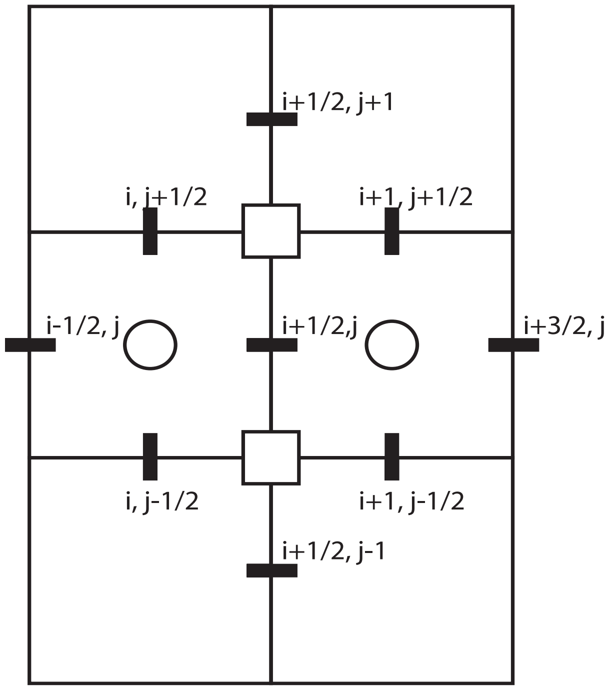

Figure 3.1: The 2D stencil for updating the velocity sample $u_{i+1/2,j}$. Circles indicate cell centred (diagonal) stress component, small squares indicate node centred (off-diagonal) stress components, and dashes indicate velocity components.

The discrete x-direction velocity components $\tilde{u}$ and the y-direction velocity components $\tilde{v}$ of the Laplacian form can be updated separately. Suppose the vector of discrete shear stress tensors for updating $\tilde{u}$ is ordered in the form:

$$\boldsymbol{\tau_x} = \begin{bmatrix} \tilde{\tau}_{xx} \\ \tilde{\tau}_{xy} \end{bmatrix}, \tag{3.11}$$

where $\tilde{\tau}_{xx}$ and $\tilde{\tau}_{xy}$ are vectors of discrete stress components and the over tilde notation is used to indicate the vectors of discrete samples corresponding to their continuous fields. The central finite difference approximations of the $\tilde{\tau}_{xx}$ at $[i, j]$ (left grid cell centre in Figure 3.1), and the $\tilde{\tau}_{xy}$ at $[i + 1/2, j + 1/2]$ (top node shared by two grid cells in Figure 3.1) are

$$\begin{aligned} (\tilde{\tau}_{xx})_{i,j} &= \mu_{i,j} \frac{\tilde{u}_{i+1/2,j} - \tilde{u}_{i-1/2,j}}{\Delta x}, \\ (\tilde{\tau}_{xy})_{i+1/2,j+1/2} &= \mu_{i+1/2,j+1/2} \frac{\tilde{u}_{i+1/2,j+1} - \tilde{u}_{i+1/2,j}}{\Delta x}. \end{aligned} \tag{3.12}$$

According to (3.2) (by replacing the deformation rate operator with the gradient operator for the Laplacian form), we have the expression for the discrete shear stress tensors:

$$\boldsymbol{\tau_x} = \mathbf{M}_{\boldsymbol{\tau}_\mathbf{x}} \mathbf{G}_{\tilde{\mathbf{u}}} \tilde{u}, \tag{3.13}$$

where $\mathbf{M}_{\boldsymbol{\tau_x}}$ is a diagonal matrix of viscosity coefficients per $\boldsymbol{\tau_x}$ sample and $\mathbf{G}_{\tilde{\mathbf{u}}}$ is the discrete gradient operator for the x-direction velocity components (i.e., $\mathbf{G}\tilde{\mathbf{u}} \approx \boldsymbol{\nabla} u$). Since the Laplacian form only works properly when the viscosity of the fluid is constant, we can use a constant μ to replace $\mathbf{M}_{\boldsymbol{\tau_x}}$. Equation 3.13 is thus simplified to

$$\boldsymbol{\tau_x} = \mu \mathbf{G}_{\tilde{\mathbf{u}}} \tilde{u}. \tag{3.14}$$

If we move one step further and check the velocity component update as described in (3.8) (by replacing the deformation rate operator with the gradient operator again), we obtain the linear system for $\tilde{u}$ given by the Laplacian form:

$$(\mathbf{P}_{\tilde{\mathbf{u}}} + \Delta t \mu \mathbf{G}_{\tilde{\mathbf{u}}}^{\mathbf{T}} \mathbf{G}_{\tilde{\mathbf{u}}}) \tilde{u} = \mathbf{P}_{\tilde{\mathbf{u}}} \tilde{u}^{old}, \tag{3.15}$$

where $\mathbf{P}_{\tilde{\mathbf{u}}}$ is a diagonal matrix of fluid densities per $\tilde{u}$ sample and $\mathbf{G}_{\tilde{\mathbf{u}}}^{\mathbf{T}}$ is the transpose of the discrete gradient operator. This is the discrete version of (3.5). It's obvious that this system is SPD since $\mathbf{P}_{\tilde{\mathbf{u}}}$ is diagonal and $\mathbf{G}_{\tilde{\mathbf{u}}}^{\mathbf{T}} \mathbf{G}_{\tilde{\mathbf{u}}}$ is SPD. One trick we apply here is using the negative transpose of the discrete (scalar) gradient operator to substitute for the discrete (vector) divergence operator

$$\mathbf{V}_{\tilde{\mathbf{u}}} = -\mathbf{G}_{\tilde{\mathbf{u}}}^{\mathbf{T}}. \tag{3.16}$$

This convenient relationship between the finite difference gradient and divergence operators on the MAC grid mirrors an integration-by-parts relationship possessed by their continuous counterparts, and is what ensures the system is SPD. This property has been used in various methods, such as the low-order discrete calculus method [Perot and Subramanian, 2007]. It is also similar to the way of generating the gradient operator in the mimetic finite difference method [Lipnikov et al., 2014], which defines a discrete finite-volume-style divergence operator, constructs a discrete inner product satisfying certain requirements, and gets a gradient operator as the negative adjoint of the divergence one. It is straightforward to derive the corresponding linear system for $\tilde{v}$ by replacing all symbols with $\tilde{v}$-related ones in (3.15).

The discretization of the shear stress tensor of the full form is different than that of the Laplacian form since they involve terms that couple the different components of velocity. The expressions for $\tilde{\tau}_{xx}$, $\tilde{\tau}_{yy}$ at $[i,j]$, and $\tilde{\tau}_{xy}$, $\tilde{\tau}_{yx}$ at $[i+1/2, j+1/2]$ become

$$\begin{aligned}
(\tilde{\tau}_{xx})_{i,j} &= 2\mu_{i,j}\frac{\tilde{u}_{i+1/2,j}-\tilde{u}_{i-1/2,j}}{\Delta x},\\
(\tilde{\tau}_{yy})_{i,j} &= 2\mu_{i,j}\frac{\tilde{v}_{i,j+1/2}-\tilde{v}_{i,j-1/2}}{\Delta x},\\
(\tilde{\tau}_{xy})_{i+1/2,j+1/2} &= (\tilde{\tau}_{yx})_{i+1/2,j+1/2}\\
&= \mu_{i+1/2,j+1/2}\left(\frac{\tilde{u}_{i+1/2,j+1}-\tilde{u}_{i+1/2,j}}{\Delta x}+\frac{\tilde{v}_{i+1,j+1/2}-\tilde{v}_{i,j+1/2}}{\Delta x}\right).
\end{aligned} \tag{3.17}$$

Since both $\tilde{u}$ and $\tilde{v}$ samples are used to update $\tilde{\tau}_{xy}$, we can no longer solve $\tilde{u}$ and $\tilde{v}$ in two separate systems. Therefore, the vector of velocity samples of the full form is arranged as

$$\mathbf{u} = \begin{bmatrix}\tilde{u}\\ \tilde{v}\end{bmatrix}. \tag{3.18}$$

where all the x-direction velocity components are ordered ahead of the y-direction ones in $\mathbf{u}$. Similarly, the vector of shear stress tensors is arranged as:

$$\boldsymbol{\tau} = \begin{bmatrix}\tilde{\tau}_{xx}\\ \tilde{\tau}_{yy}\\ \tilde{\tau}_{xy}\end{bmatrix}. \tag{3.19}$$

Note that the off-diagonal stress tensor $\tilde{\tau}_{yx}$ is not included in $\boldsymbol{\tau}$ since it has the same expression as $\tilde{\tau}_{xy}$ (i.e., the stress tensor is symmetric). According to (3.2), $\boldsymbol{\tau}$ can be calculated by

$$\boldsymbol{\tau} = 2\mathbf{MDu}, \tag{3.20}$$

where $\mathbf{M}$ is a diagonal matrix of viscosity coefficients per stress sample and $\mathbf{D}$ is the discrete deformation rate operator for velocity (i.e., $\mathbf{Du} \approx \frac{(\boldsymbol{\nabla}\vec{u}+\boldsymbol{\nabla}\vec{u}^T)}{2}$), which is determined by the finite difference stencils given in (3.17). If the numbers of degrees of freedom for the discrete velocity and stress tensor fields are $n_{\mathbf{u}}$ and $n_{\boldsymbol{\tau}}$, respectively, then $\mathbf{D}$ is a $n_{\mathbf{u}} \times n_{\boldsymbol{\tau}}$ matrix.

The Laplacian form we described above relied on the relationship between the scalar gradient operator and the vector divergence operator to ensure symmetry, since only one velocity component is used. By contrast, symmetry in the case of the full form will depend on the existence of a relationship between the (vector) deformation rate operator and the *tensor* divergence operator. At first glance, it is not necessarily obvious that the negative adjoint relationship still holds. However, if we can prove the tensor divergence operator is equal to the negative transpose of $\mathbf{D}$ (possibly scaled by a constant or diagonal matrix), the linear system will still be SPD (which will be further discussed in Section 3.3.2) and the computation time spent on solving the system will typically be much less than it would be for a comparable non-symmetric system. This can be shown as follows:

Suppose we write the discrete gradient operators for the x- and y-direction velocity components used in the Laplacian form as

$$\mathbf{G}_{\tilde{\mathbf{u}}} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \mathbf{G}_{\tilde{\mathbf{v}}} = \begin{bmatrix} A_3 \\ A_4 \end{bmatrix}, \tag{3.21}$$

where A_1 and A_2 are discrete partial derivative operators on $\tilde{u}$ such that $A_1\tilde{u} \approx \frac{\partial u}{\partial x}$ and $A_2\tilde{u} \approx \frac{\partial u}{\partial y}$, and A_3 and A_4 are discrete partial derivative operators on $\tilde{v}$ such that $A_3\tilde{v} \approx \frac{\partial v}{\partial y}$ and $A_4\tilde{v} \approx \frac{\partial v}{\partial x}$. Then, we write the corresponding discrete divergence operators as

$$\mathbf{V}_{\tilde{\mathbf{u}}} = \begin{bmatrix} B_1 & B_2 \end{bmatrix}, \mathbf{V}_{\tilde{\mathbf{v}}} = \begin{bmatrix} B_3 & B_4 \end{bmatrix} \tag{3.22}$$

such that $B_1\tilde{\tau}_{xx} \approx \frac{\partial \tau_{xx}}{\partial x}, B_2\tilde{\tau}_{xy} \approx \frac{\partial \tau_{xy}}{\partial y}, B_3\tilde{\tau}_{yy} \approx \frac{\partial \tau_{yy}}{\partial y}$ and $B_4\tilde{\tau}_{yx} \approx \frac{\partial \tau_{yx}}{\partial x}$. Because of (3.16),

$$\begin{aligned} A_1 &= -B_1^{\mathbf{T}}, A_2 = -B_2^{\mathbf{T}}, \\ A_3 &= -B_3^{\mathbf{T}}, A_4 = -B_4^{\mathbf{T}}. \end{aligned} \tag{3.23}$$

Although the discrete stress tensors of the full form have different expressions, they sit at the same positions as their Laplacian counterparts. Hence, we can use the blocks A_1, A_2, A_3 and A_4 to construct the discrete deformation rate operator:

$$\mathbf{D} = \begin{bmatrix} A_1 & 0 \\ 0 & A_3 \\ 0.5A_2 & 0.5A_4 \end{bmatrix} \tag{3.24}$$

and create the discrete divergence operator with the blocks B_1, B_2, B_3 and B_4:

$$\mathbf{V} = \begin{bmatrix} B_1 & 0 & B_2 \\ 0 & B_3 & B_4 \end{bmatrix}. \tag{3.25}$$

Because of (3.23), the discrete divergence operator can be expressed as

$$\mathbf{V} = -\mathbf{D^T K} \tag{3.26}$$

where $\mathbf{K}$ is a diagonal matrix in the form of $\begin{bmatrix} I_{n_{\tilde{\tau}_{xx}}} & & \\ & I_{n_{\tilde{\tau}_{yy}}} & \\ & & 2I_{n_{\tilde{\tau}_{xy}}} \end{bmatrix}$. This factoring diagonal only arises in the full form of viscosity because the off-diagonal stress tensor $\tilde{\tau}_{yx}$ is ignored in (3.19) since τ is symmetric. Suppose we include $\tilde{\tau}_{yx}$ in the system; the vector of the stress tensors becomes

$$\boldsymbol{\tau'} = \begin{bmatrix} \tilde{\tau}_{xx} \\ \tilde{\tau}_{yy} \\ \tilde{\tau}_{xy} \\ \tilde{\tau}_{yx} \end{bmatrix}.$$

The discrete deformation rate operator and the discrete divergence operator turn into

$$\mathbf{D}' = \begin{bmatrix} A_1 & 0 \\ 0 & A_3 \\ 0.5A_2 & 0.5A_4 \\ 0.5A_2 & 0.5A_4 \end{bmatrix}$$

and

$$\mathbf{V}' = \begin{bmatrix} B_1 & 0 & 0.5B_2 & 0.5B_2 \\ 0 & B_3 & 0.5B_4 & 0.5B_4 \end{bmatrix},$$

respectively. It is obvious that $\mathbf{V}' = -(\mathbf{D}')^{\mathbf{T}}$.

Then the full form variant of (3.15) can be written as

$$(\mathbf{P} + 2\Delta t \mathbf{D^T KMD})\mathbf{u} = \mathbf{Pu^{old}}. \tag{3.27}$$

where $\mathbf{P}$ is a diagonal matrix of densities per velocity sample, $\mathbf{M}$ is a diagonal matrix of viscosity coefficients per stress sample, $\mathbf{D}$ and $\mathbf{K}$ are the discrete deformation rate operator and the factoring matrix discussed before. Note that this simple finite difference form doesn't yet incorporate free surface boundaries. Hence, we move on to adopting the variational form in Section 3.3.2.

From now on, we simply use the symbol $-\mathbf{D^T K}$ to represent the discrete divergence operator because (1) we do use (3.26) to compute the divergence operator in the system; and (2) this form makes it more obvious that the linear system is SPD.

3.3.2 Discretization of the Variational Form

We have introduced the energy functional 3.10 and discussed the fact that optimizing it leads to a valid solution to our original PDE. Here we discretize it in space by approximating $E[\vec{u}]$ with sums and finite differences on a staggered grid, scaling the contributions of each cell by the volume of cell and the fraction of the cell that represents liquid. Although this simple discretization strategy provides only first order accuracy, it is nevertheless sufficient to preserve all the interesting visual behavior of viscous flow. For example, the first integral of (3.10) becomes (we consider only the 2D case in this thesis)

$$\begin{aligned}\iint_\Omega \rho\left\|\vec{u}-\vec{u}^{old}\right\|^2 &= \iint_\Omega \rho(u-u^{old})^2 + \iint_\Omega \rho(v-v^{old})^2 \\ &\approx \sum_{(i+1/2,j)\in\Omega} k_{i+1/2,j}\rho_{i+1/2,j}(u_{i+1/2,j}-u^{old}_{i+1/2,j})^2\Delta x^2 \\ &+ \sum_{(i,j+1/2)\in\Omega} k_{i,j+1/2}\rho_{i,j+1/2}(v_{i,j+1/2}-v^{old}_{i,j+1/2})^2\Delta x^2 \quad ,\end{aligned} \tag{3.28}$$

where k is a factor indicating the fraction of a cell covered by fluid and has a value of 1 for velocity samples whose control volumes are entirely inside the fluid domain. The subscript of k is the index of the corresponding control volume. A detailed explanation of the control volume calculation for cells near the liquid surface is provided in Section 3.4.3. At this stage, we just need to know k is piecewise constant within each cell and can vary across the grid.

Similarly, we can break down the second integral into several components by expanding the Frobenius norm:

$$2\Delta t \iint_\Omega \mu \left\| \frac{\nabla\vec{u}+\nabla\vec{u}^T}{2} \right\|_F^2 = 2\Delta t \iint_\Omega \mu\left(u_x^2+\frac{1}{2}(u_y+v_x)^2+v_y^2\right). \tag{3.29}$$

Each partial derivative can be approximated by a central finite difference. For example,

$$\begin{aligned}&2\Delta t\iint_\Omega \mu u_x^2 \approx 2\Delta t\sum_{i,j} k_{i,j}\mu_{i,j}\left(\frac{\tilde{u}_{i+1/2,j}-\tilde{u}_{i-1/2,j}}{\Delta x}\right)^2\Delta x^2, \\ &2\Delta t\iint_\Omega \mu\frac{1}{2}(u_y+v_x)^2 \\ &\approx \Delta t\sum_{i,j} k_{i+1/2,j+1/2}\mu_{i+1/2,j+1/2}\left(\frac{\tilde{u}_{i+1/2,j+1}-\tilde{u}_{i+1/2,j}+\tilde{v}_{i+1,j+1/2}-\tilde{v}_{i,j+1/2}}{\Delta x}\right)^2\Delta x^2.\end{aligned} \tag{3.30}$$

Therefore, the discrete energy functional has the following form:

$$
\begin{aligned}
E[\mathbf{u}] = & \sum_{i,j} k_{i+1/2,j}\rho_{i+1/2,j}(\tilde{u}_{i+1/2,j} - \tilde{u}^{old}_{i+1/2,j})^2\Delta x^2 \\
& + \sum_{i,j} k_{i,j+1/2}\rho_{i,j+1/2}(\tilde{v}_{i,j+1/2} - \tilde{v}^{old}_{i,j+1/2})^2\Delta x^2 \\
& + 2\Delta t \sum_{i,j} k_{i,j}\mu_{i,j}(\tilde{u}_{i+1/2,j} - \tilde{u}_{i-1/2,j})^2 + (\tilde{v}_{i,j+1/2} - \tilde{v}_{i,j-1/2})^2 \\
& + \Delta t \sum_{i,j} k_{i+1/2,j+1/2}\mu_{i+1/2,j+1/2}(\tilde{u}_{i+1/2,j+1} - \tilde{u}_{i+1/2,j} + \tilde{v}_{i+1,j+1/2} - \tilde{v}_{i,j+1/2})^2
\end{aligned}
\tag{3.31}
$$

By differentiating the above equation with respect to each unknown velocity, setting the gradient to zero and combining all resultant equations together, we get a linear system to solve for the new velocities. Here we give an example of differentiating $E[\mathbf{u}]$ with respect to $\tilde{u}_{i+1/2,j}$ which corresponds to one row of the final linear system.

$$
\begin{aligned}
& k_{i+1/2,j}\rho_{i+1/2,j}(\tilde{u}_{i+1/2,j} - \tilde{u}^{old}_{i+1/2,j})\Delta x^2 \\
& + 2\Delta t k_{i,j}\mu_{i,j}\left(\frac{\tilde{u}_{i+1/2,j} - \tilde{u}_{i-1/2,j}}{\Delta x^2}\right)\Delta x^2 \\
& - 2\Delta t k_{i+1,j}\mu_{i+1,j}\left(\frac{\tilde{u}_{i+3/2,j} - \tilde{u}_{i+1/2,j}}{\Delta x^2}\right)\Delta x^2 \\
& + \Delta t k_{i+1/2,j-1/2}\mu_{i+1/2,j-1/2}\left(\frac{\tilde{u}_{i+1/2,j} - \tilde{u}_{i+1/2,j-1} + \tilde{v}_{i+1,j-1/2} - \tilde{v}_{i,j-1/2}}{\Delta x^2}\right)\Delta x^2 \\
& - \Delta t k_{i+1/2,j+1/2}\mu_{i+1/2,j+1/2}\left(\frac{\tilde{u}_{i+1/2,j+1} - \tilde{u}_{i+1/2,j} + \tilde{v}_{i+1,j+1/2} - \tilde{v}_{i,j+1/2}}{\Delta x^2}\right)\Delta x^2 = 0.
\end{aligned}
\tag{3.32}
$$

Substituting (3.17) in (3.32), we reach

$$
\begin{aligned}
& w_{i+1/2,j}\rho_{i+1/2,j}(\tilde{u}_{i+1/2,j} - \tilde{u}^{old}_{i+1/2,j}) \\
& + \Delta t w_{i,j}\mu_{i,j}\frac{(\tilde{\tau}_{xx})_{i,j}}{\Delta x} - \Delta t w_{i+1,j}\mu_{i+1,j}\frac{(\tilde{\tau}_{xx})_{i+1,j}}{\Delta x} \\
& + \Delta t w_{i+1/2,j-1/2}\mu_{i+1/2,j-1/2}\frac{(\tilde{\tau}_{xy})_{i+1/2,j-1/2}}{\Delta x} \\
& - \Delta t w_{i+1/2,j+1/2}\mu_{i+1/2,j+1/2}\frac{(\tilde{\tau}_{xy})_{i+1/2,j+1/2}}{\Delta x} = 0.
\end{aligned}
\tag{3.33}
$$

where $w_{i,j}$ is equal to $k_{i,j}\Delta x^2$ and represents the amount of fluid in the cell $[i, j]$. We know the finite difference approximation of the divergence operator at $[i+1/2, j]$ is

$$(V\tilde{\tau})_{i+1/2,j} = \frac{(\tilde{\tau}_{xx})_{i+1,j} - (\tilde{\tau}_{xx})_{i,j} + (\tilde{\tau}_{xy})_{i+1/2,j+1/2} - (\tilde{\tau}_{xy})_{i+1/2,j-1/2}}{\Delta x},$$

where V is a local discrete divergence operator. Then, (3.33) can be written as

$$w_{i+1/2,j}\rho_{i+1/2,j}(\tilde{u}_{i+1/2,j} - \tilde{u}^{old}_{i+1/2,j}) - \Delta t(VMW\tilde{\tau})_{i+1/2,j} = 0. \tag{3.34}$$

Here M and W are both 2 by 2 diagonal matrices, and represent the viscosity coefficients and the control volumes of local stress tensors, respectively. By assembling equations with respect to all unknown velocities and substituting the discrete divergence operator with (3.26), we have

$$(\mathbf{W_u P} + 2\Delta t\mathbf{D^T KMW_{\boldsymbol{\tau}} D})\mathbf{u} = \mathbf{W_u P u^{old}}, \tag{3.35}$$

where $\mathbf{W_u}$ and $\mathbf{W_{\boldsymbol{\tau}}}$ are the control volumes of the velocities and stress tensors, $\mathbf{P}$ is a diagonal matrix of densities per velocity sample, $\mathbf{M}$ is a diagonal matrix of viscosity coefficients per stress sample, $\mathbf{D}$ and $\mathbf{K}$ are the discrete deformation rate operator and the factoring matrix discussed in Section 3.3.1. The linear system of (3.35) is SPD since $\mathbf{W_u}, \mathbf{W_{\boldsymbol{\tau}}}, \mathbf{P}, \mathbf{M}, \mathbf{K}$ are all diagonal matrices. Note that the way we construct the linear system is very similar to the Rayleigh-Ritz method widely used in finite element analysis [Bathe and Wilson, 1976].

Another way to get (3.35) is to rewrite $E[\mathbf{u}]$ as

$$E[\mathbf{u}] = (\mathbf{u} - \mathbf{u^{old}})^\mathbf{T}\mathbf{W_u P}(\mathbf{u} - \mathbf{u^{old}}) + 2\Delta t\mathbf{u^T D^T KMW_{\boldsymbol{\tau}} Du}. \tag{3.36}$$

The diagonal K is included due to the discrepancy between the continuous and discrete Frobenius norm (i.e., $\left\|\frac{\nabla\vec{u}+\nabla\vec{u}^T}{2}\right\|_F^2 = \tau_{xx}^2 + 2\tau_{xy}^2 + \tau_{yy}^2$ whereas $(\mathbf{Du})^\mathbf{T}(\mathbf{Du}) = \tilde{\tau}_{xx}^2 + \tilde{\tau}_{xy}^2 + \tilde{\tau}_{yy}^2$). Differentiating $E[\mathbf{u}]$ with respect to $\mathbf{u}$ gives us (3.35). This approach does not require the explicit use of the negative adjoint relationship between the discrete deformation rate and divergence operators (as a matter of fact, it doesn't require a divergence operator be defined at all), and automatically yields a SPD system.

3.4 Adaptive Grids

We have reviewed and derived the regular grid viscosity solver of Batty and Bridson [2008]. In this section, we will discuss how to extend this discretization and control volume calculations to the quadtree grid, and emphasize the difference from the regular grid. Figure 3.2

illustrates two different quadtree grids. The left one is categorized as a graded tree since there is at most one level difference in resolution between neighbouring cells. Note that the cells connected only by a grid node (i.e., diagonal to each other) are also considered as neighbouring cells since each off-diagonal stress tensor sits at a grid node and the four surrounding velocities are used to update it. The right grid in Figure 3.2 is a non-graded quadtree because there is no restriction on the level difference between neighbouring cells. Based on these definitions, we can consider the graded tree grid as a special case of the non-graded tree grid. For simplicity, we only consider the graded quadtree in this thesis. Handling the non-graded case may be an interesting direction for future work.

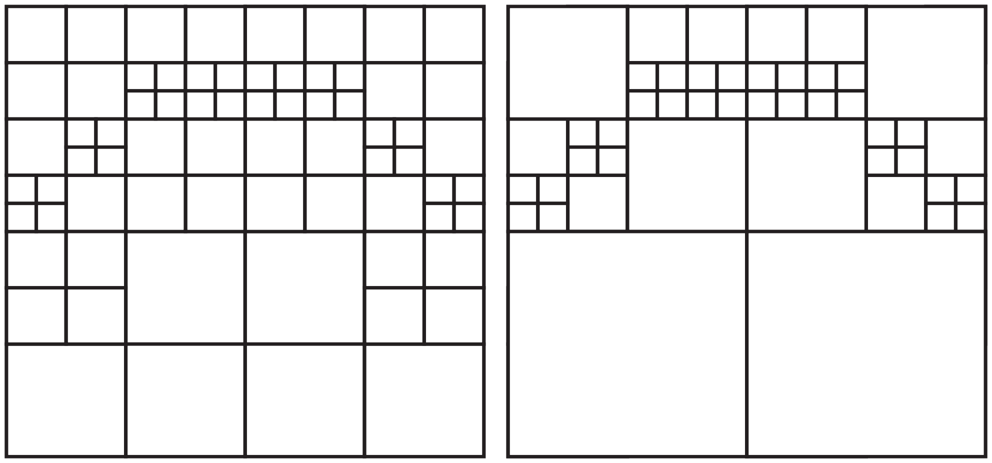
Figure 3.2: Graded (left) and non-graded (right) quadtree grids.

3.4.1 Discretization

Figure 3.3 shows an example in which cells at different levels are adjacent to each other, where we once again indicate the locations of velocity and stress components in the same manner as Figure 3.1. We call it a T-junction as its shape suggests, and only store a velocity sample for the large face and ignore the small faces at these intersections. Another way to look at it is assuming that the small faces both have the same velocity as the big face as shown in Figure 3.3. We make this choice for simplicity and consistency with the octree pressure projection method of [Losasso et al., 2006a]. With this assumption in mind, we will develop a discretization of the deformation rate operator in the presence of T-junctions, and as in the preceding section, the corresponding tensor divergence will be defined (implicitly) by its transpose.

Discrete diagonal stress components $\tilde{\tau}_{xx}$ and $\tilde{\tau}_{yy}$ next to a T-junction are calculated by

$$\begin{aligned} \tilde{\tau}_{xx} &= 2\mu\frac{u_1 - u_2}{h}, \\ \tilde{\tau}_{yy} &= 2\mu\frac{v_1 - v_2}{h}, \end{aligned} \tag{3.37}$$

respectively, where u_1, u_2, v_1, v_2 and h are all illustrated in Figure 3.3. Note that the stencil for the large cell's cell-centred data does not have to change as shown in Figure 3.4, since the big face velocity already exists.

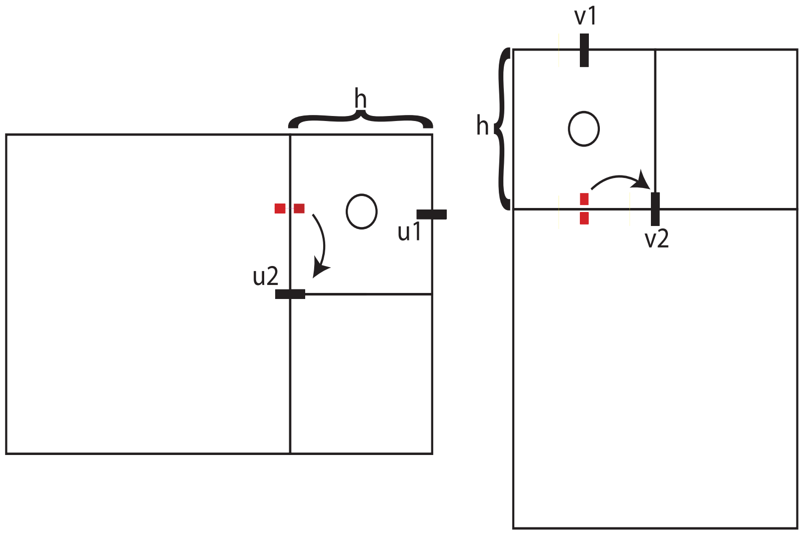

Figure 3.3: Stencils for $\tilde{\tau}_{xx}$ (left) and $\tilde{\tau}_{yy}$ (right) at a T-junction of an adaptive grid. Arrows indicate that (ghost) small faces take their velocity value from the neighbouring large face velocity sample.

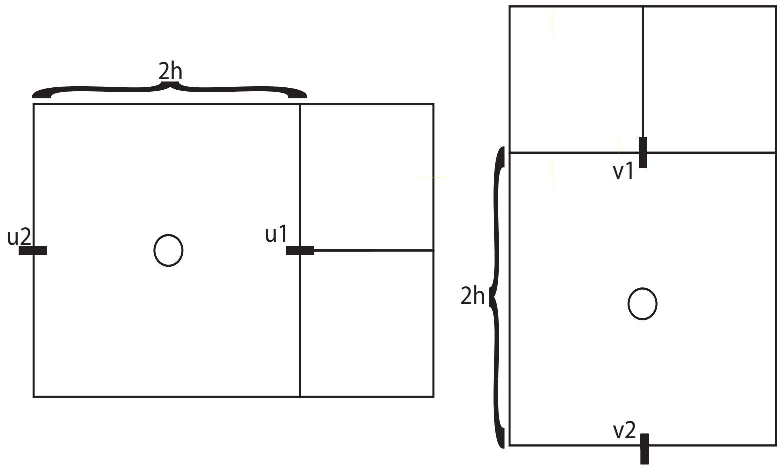

Figure 3.4: Stencils for $\tilde{\tau}_{xx}$ (left) and $\tilde{\tau}_{yy}$ (right) for the large cell at a T-junction of an adaptive grid.

The off-diagonal stress component at the T-junction is much more difficult to compute. As given in (3.38) and illustrated in Figure 3.5, an off-diagonal stress component on a regular grid is calculated from the four velocities around it: two verticals and two horizontals.

$$\tilde{\tau}_{xy} = \mu \left(\frac{u_1 - u_2}{h} + \frac{v_1 - v_2}{h} \right) \tag{3.38}$$

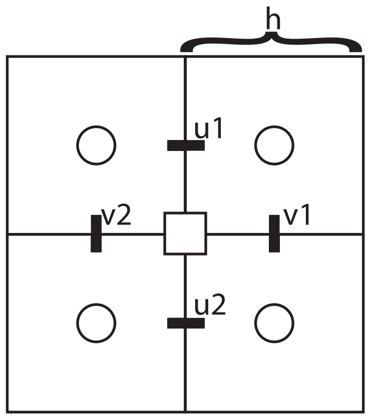

Figure 3.5: Stencil for $\tilde{\tau}_{xy}$ on a regular grid.

In order to perform the corresponding calculation at a T-junction on an adaptive grid, we have to bring in more velocity samples as illustrated in Figure 3.6.

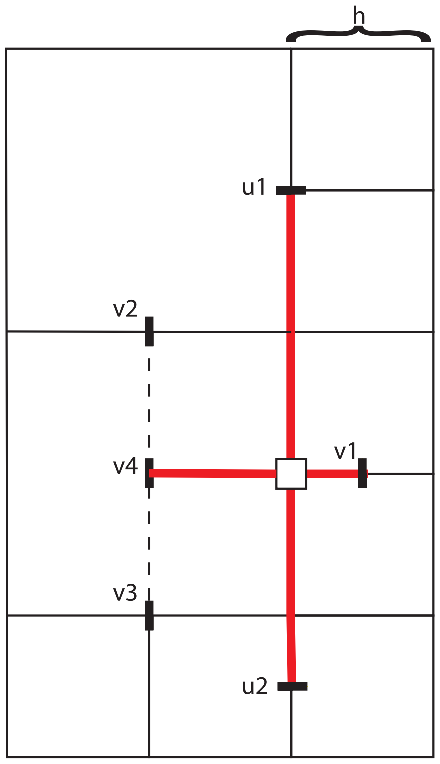

Figure 3.6: Stencil for $\tilde{\tau}_{xy}$ at a T-junction on an adaptive grid.

As compared to the regular grid case where the immediately neighbouring velocities are used in the stencil, it is obvious that only the velocity v_1 can be used in the discretization directly. v_4 does not exist on the grid and is introduced as an average of v_2 and v_3. In addition, we have to bring in u_1 and u_2 which are above and below the cell, respectively, because we assume a piecewise constant velocity on the entire T-junction face (which implies there is no gradient within the face itself). The off-diagonal stress component for this specific geometric scenario is calculated by

$$\tilde{\tau}_{xy} = \mu \left(\frac{u_1 - u_2}{3.5h} + \frac{v_1 - 0.5v_2 - 0.5v_3}{1.5h} \right), \tag{3.39}$$

where all symbols are illustrated in Figure 3.6. The coefficients given above for the v-components will hold at all such (horizontal) T-junctions, since we only deal with a graded quadtree grid. But this is not the case for u-components: the actual coefficients are dependent on the resolution of the cells above and below. Since this is no longer exactly a central finite difference, the accuracy reduces to first-order.

Notice that since the variational form implicitly enforces the symmetry relationship between deformation rate and divergence, we only need to describe and construct the deformation rate stencil on the quadtree grid; the corresponding divergence stencil is given automatically as in the regular grid case.

3.4.2 Control Volumes of Internal Fluid

So far we have discussed the viscosity coefficient and operators in (3.35); the only remaining terms are control volumes $\mathbf{W_u}$ and $\mathbf{W_\tau}$. Since staggered grids are used in this thesis, vertical velocity, horizontal velocity, diagonal and off-diagonal stress components are located at different positions, as illustrated in Figure 3.7 for the uniform grid case.

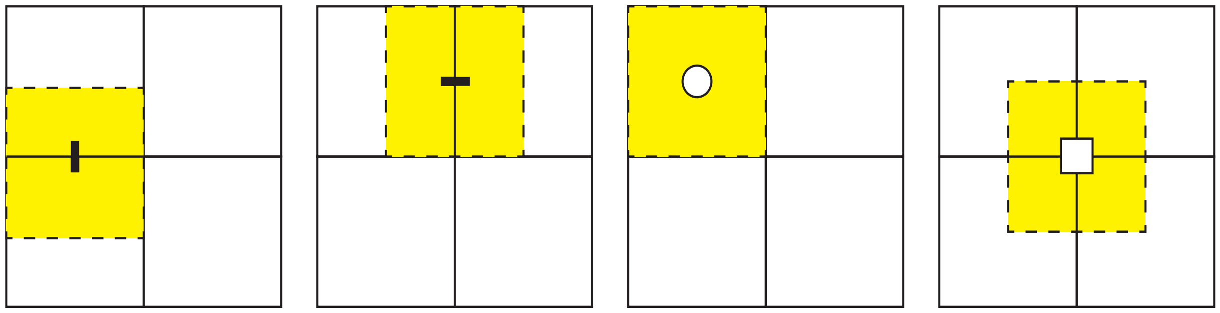
Figure 3.7: Control volumes around each sample location in 2D. From left to right: vertical velocity control volume, horizontal velocity control volume, diagonal stress control volume, off-diagonal stress control volume.

The size of the control volumes for these variables on uniform staggered grids are equal to one grid cell area inside the fluid. Because of that, we can get rid of the control volume matrix on both sides of (3.35) and get the following equation for the inner body of a fluid:

$$(\mathbf{P} + 2\Delta t \mathbf{D^T K M D})\mathbf{u} = \mathbf{P u^{old}}. \tag{3.40}$$

This is simply the discrete version of (3.8). However, we cannot eliminate the control volume matrix on an adaptive grid. We will discuss the control volumes for vertical and horizontal velocities, diagonal stresses and off-diagonal stress components on an adaptive grid separately.

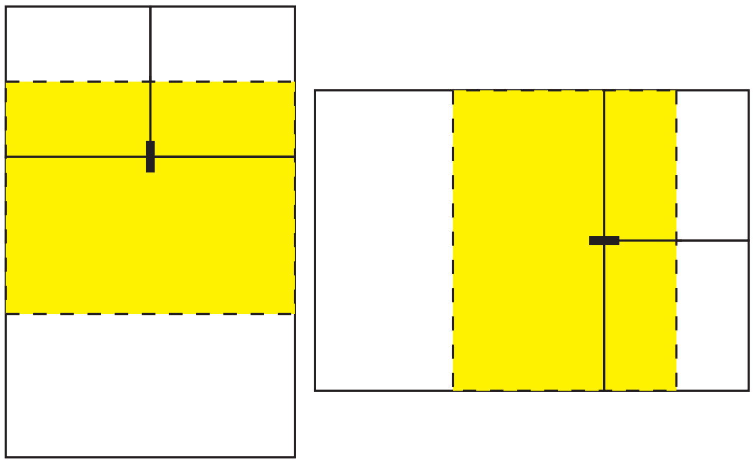

Figure 3.8: Vertical and horizontal velocity control volumes near a T-junction.

As mentioned before, we only store the large face velocity sample at a T-junction. From now on, we denote the area of a small cell as Δ. Then, the velocity control volume, shown in Figure 3.8, would have an area of 3Δ.

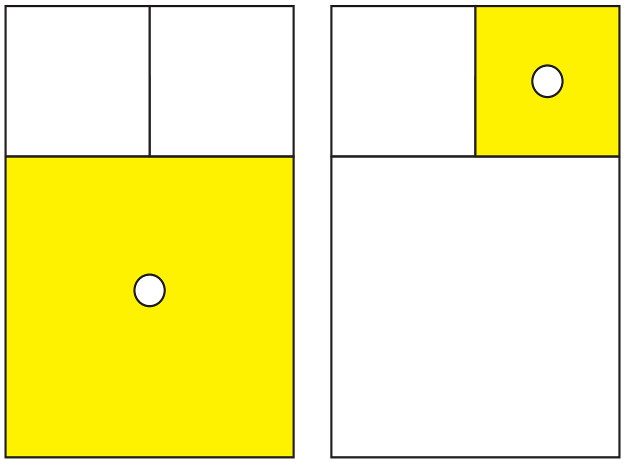

Figure 3.9: Diagonal stress control volumes on large and small grid cells.

The control volume of a diagonal stress component is the same as that on a regular grid, which is equal to the size of the grid cell that contains it. Two examples are given in Figure 3.9.

The off-diagonal stress control volume is tricky since it has several different scenarios and because we would prefer that the control volumes for a particular degree-of-freedom type not overlap. When it has only one fine cell around, the control volume area is 2.25Δ as depicted in Figure 3.10.

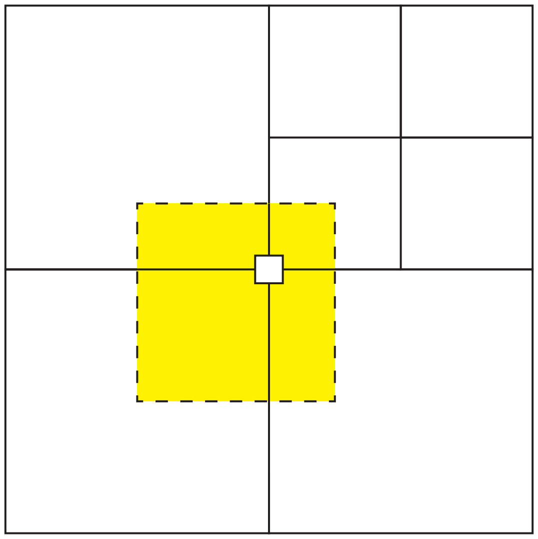

Figure 3.10: Off-diagonal stress component control volumes.

When it sits at a T-junction or has two refined cells on top as shown in Figure 3.11, the control volume area is equal to 1.5Δ.

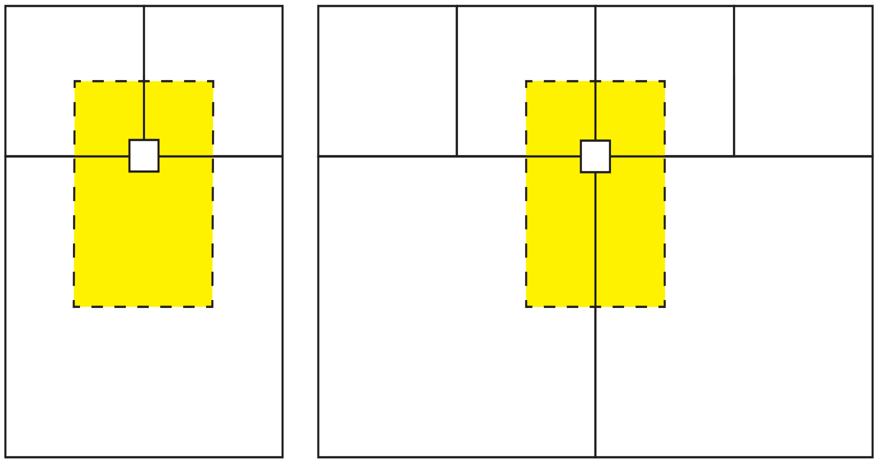
Figure 3.11: Off-diagonal stress component control volumes.

When the off-diagonal stress component has two fine cells sharing only a vertex or is surrounded by three fine cells as in Figure 3.12, the control volume area is simply the fine cell area Δ.

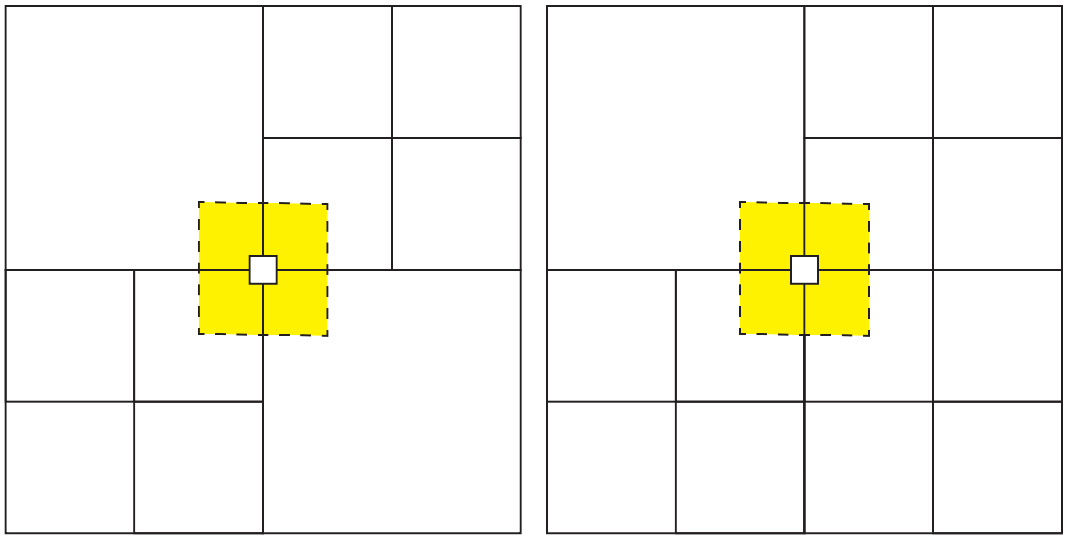
Figure 3.12: Off-diagonal stress control volumes.

One subtle case arises in which two control volumes can potentially overlap. For example, Figure 3.13 presents one scenario. We have tried several different methods to address this problem, such as assigning the overlapped area to one tensor as shown in Figure 3.13, evenly distributing the area to two tensors as shown in Figure 3.14 or simply ignoring it and using 1.5Δ for both the stresses as shown in Figure 3.15. All three methods produce similar and reasonable results and have the same convergence rate in our experiments. To simplify the implementation, we choose the third method and don't tackle the overlapped area explicitly.

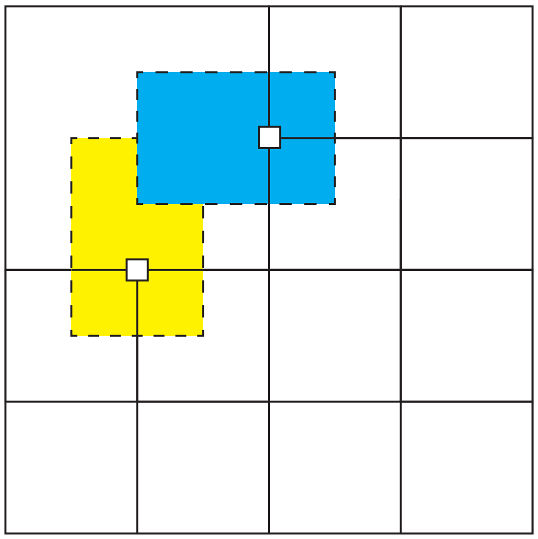

Figure 3.13: The overlapped area assigned to the top stress.

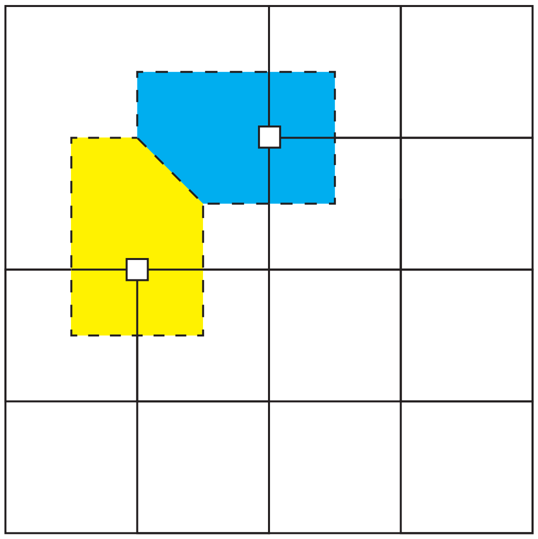

Figure 3.14: The overlapped area evenly assigned to two stresses.

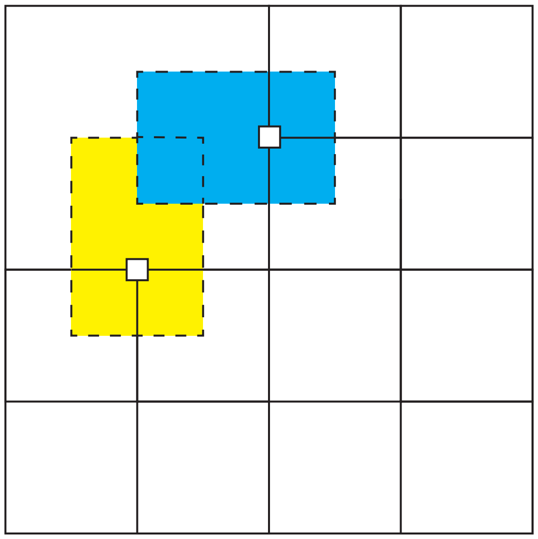

Figure 3.15: The overlapped area assigned to both stresses.

3.4.3 Control Volumes of Surface Fluid

The control volume of the variational formulation was originally brought in to correctly address the free surface boundary condition [Batty and Bridson, 2008]. Since we only start merging cells (to build the quadtree grid) when they are completely covered by the fluid, the cells on the free surface are simply the regular grid cells and we never have to deal with the free surface and the T-junction at the same time.

At the free surface, only a part of a cell is covered by the fluid and we need to approximate its area. First, assuming we possess signed distance values for the surface geometry stored on a regular grid, we evenly divide the cell into several sub-regions and compute the signed distance at each sub-region centre using bi-linear interpolation in 2D or tri-linear interpolation in 3D. We categorize the sub-regions into two groups based on the sign of the signed distance: fluid region and air region, and assume the fluid region is completely covered by fluid and the air region is filled in with air. For example, in Figure 3.16, the regions 1, 2, 3, 4, 5 are air regions because their centres have negative signed distance, and the regions 6, 7, 8, 9 are fluid regions since the signed distances at these centres are positive. So the control volume area of this cell is $\frac{4}{9}\Delta$ where Δ is the cell area. Although this subsampling approach is a very simple approximation, we found that it generates reasonable results. Another possible alternative would be to use marching squares [Lorensen and Cline, 1987] to construct a polygonal surface approximation from the distance field within each cell, and compute the exact area of each polygon.

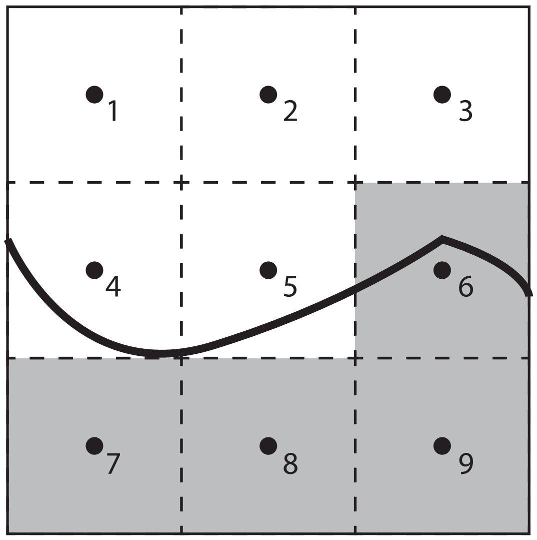

Figure 3.16: A cell on the surface of the fluid, the bold curve represents the actual fluid surface, the gray sub-regions are fluid regions and the white sub-regions are air regions.

3.5 Velocity Mapping

For the purpose of this thesis we are focused on adaptive discretization of viscosity alone. Hence to simplify the implementation effort, we only solve the viscosity step on an adaptive grid, and solve everything else on a standard regular grid. Therefore, we need to convert the velocities on a regular grid to those on an adaptive grid before the viscosity solve, and map the resultant velocities on the adaptive grid back to the regular grid. A FLIP-like approach is utilized to transfer velocity data between the two grid structures. The approach we adopt was first introduced by Schroeder et al. [2012] to convert between Eulerian and Lagrangian velocities, and applied by Batty and Houston [2011][1] to map between node- and face-velocities on tetrahedral meshes.

FLIP uses particles to advect velocities and efficiently computes all forces (pressure, viscosity and external forces) on an auxiliary grid. It increments particle velocities with the change in velocities that occurred on the grid. Compared to the earlier particle-in-cell

[1][Schroeder et al., 2012] was published later than this paper due to the long review process of a journal paper, and this paper did cite the original version of [Schroeder et al., 2012].

(PIC) method [Harlow, 1962] which simply replaces particle velocities with interpolated values of grid velocities, the FLIP method dramatically reduced the numerical dissipation [Brackbill and Ruppel, 1986]. In any case, the similarity lies in the fact that we wish to convert our velocity field and forces among two distinct representations.

If we denote the particle and grid velocities used in FLIP as $\mathbf{u}$ and $\hat{\mathbf{u}}$, respectively, then the mapping from $\mathbf{u}$ to $\hat{\mathbf{u}}$ can be expressed as:

$$\hat{\mathbf{u}} = \mathbf{H}\mathbf{u}, \tag{3.41}$$

where $\mathbf{H}$ is some linear interpolation operator. Schroeder et al. [2012] proved that a corresponding force $\mathbf{f}$ on particles has the form:

$$\mathbf{f} = \mathbf{H}^{\mathbf{T}}\hat{\mathbf{f}}, \tag{3.42}$$

where $\hat{\mathbf{f}}$ is the force computed on the grid and $\mathbf{H}^{\mathbf{T}}$ is the transpose of $\mathbf{H}$ determined in (3.41).

Schroeder et al. [2012] also stated that "in general, whenever $\mathbf{H}$ is an interpolation operator from one set of degrees of freedom to another, the operator $\mathbf{H}^{\mathbf{T}}$ can be used to distribute forces from the second set of degrees of freedom to the first". Therefore, we can use (3.41) and (3.42) directly to map between velocities on the regular and the adaptive grid. The only difference is that $\mathbf{u}$ and $\mathbf{f}$ turn into the variables on the regular grid and $\hat{\mathbf{u}}$ and $\hat{\mathbf{f}}$ become variables on the adaptive grid.

Figure 3.17 illustrates a stencil for a single level mapping for x-axis velocities. The velocities u_1 to u_6 are the values on a fine grid and u' represents an unknown velocity on a coarse grid. The way to compute u' is given by

$$u' = 0.125u_1 + 0.25u_2 + 0.125u_3 + 0.125u_4 + 0.25u_5 + 0.125u_6, \tag{3.43}$$

where the interpolation coefficients are inversely proportional to the distance between the fine and coarse velocity values. Note that (3.43) is corresponding to a row of the mapping matrix $\mathbf{H}$, so we can construct $\mathbf{H}$ by assembling equations with respect to all unknown coarse velocities.

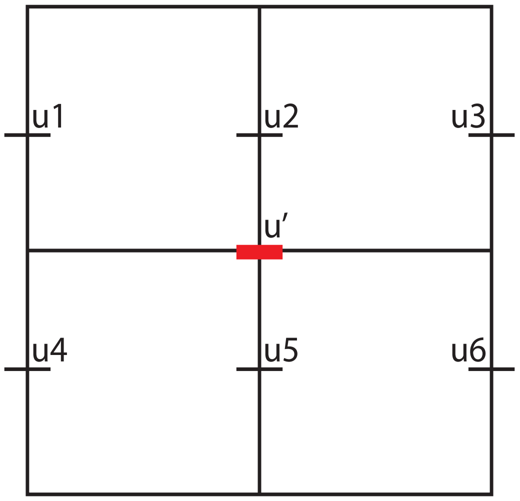

Figure 3.17: Mapping stencil for getting a coarse x-axis velocity with surrounding fine velocity values. The thick red and the thin black dashes represent the fine and coarse velocities, respectively.

Since a quadtree may contain more than 2 levels, we need to calculate interpolation operators between each two adjacent levels and multiply them together to obtain the **H** matrix. Figures 3.18, 3.19 and 3.20 illustrate how we create a 3-layer quadtree grid from a regular one, assuming that the finest refinement level is along the right boundary, and the remainder of the domain is to be coarsened.

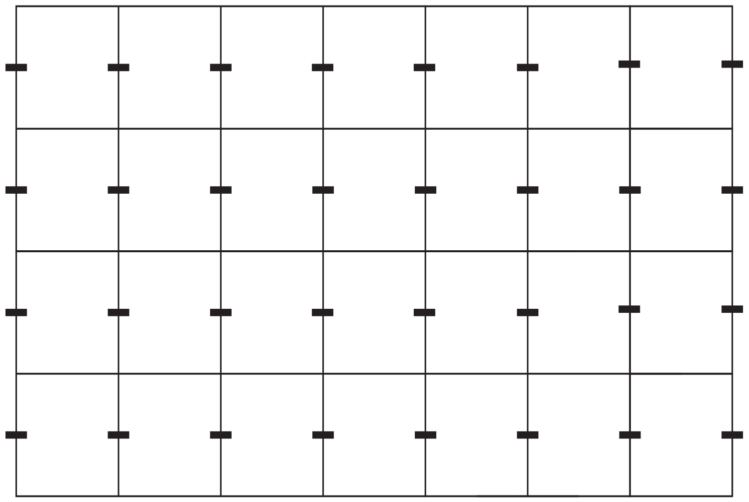

Figure 3.18: Regular grid.

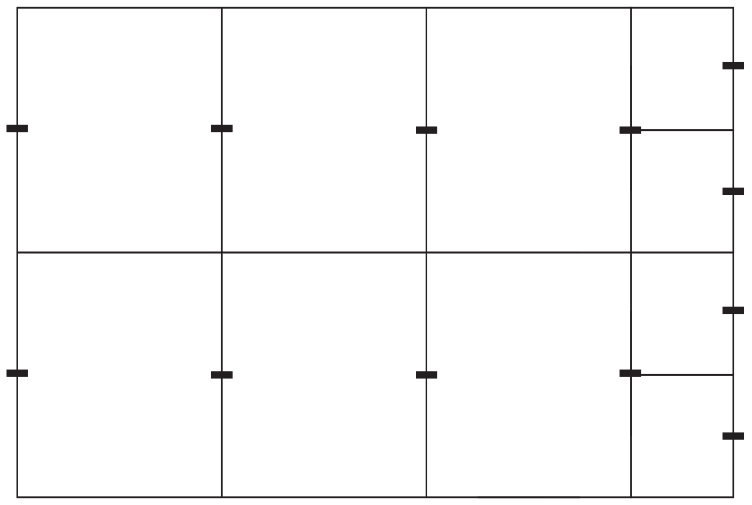

Figure 3.19: Intermediate grid after one level of coarsening.

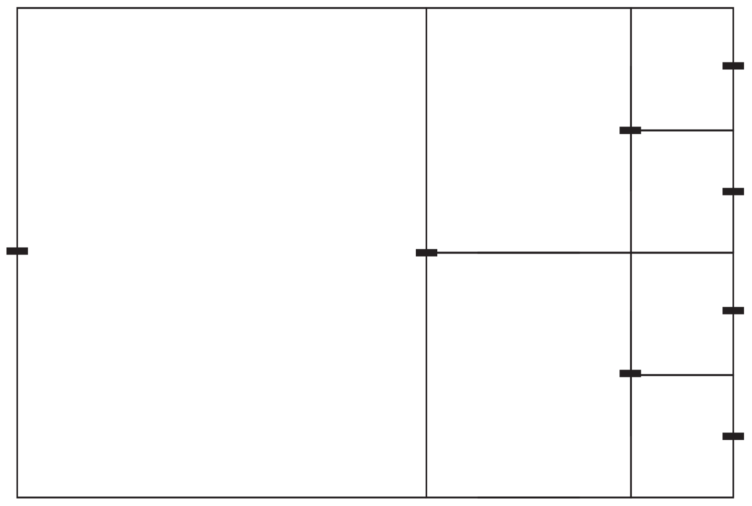

Figure 3.20: Final quadtree grid after two levels of coarsening.

By substituting our new symbols in (3.35) and changing the order of the terms, we have

$$\mathbf{W}_{\hat{\mathbf{u}}}\mathbf{P}\hat{\mathbf{u}} = \mathbf{W}_{\hat{\mathbf{u}}}\mathbf{P}\hat{\mathbf{u}}^{\mathbf{old}} - 2\Delta t\mathbf{D}^{\mathbf{T}}\mathbf{K}\mathbf{M}\mathbf{W}_{\hat{\boldsymbol{\tau}}}\mathbf{D}\hat{\mathbf{u}}. \tag{3.44}$$

The second term on the right hand side of (3.44) is the integral of the viscous force over a cell on the adaptive grid. By applying (3.42), we derive the method of mapping the velocity change back to the regular grid:

$$\mathbf{W_u P u} = \mathbf{W_u P u^{old}} + 2\Delta t \mathbf{H^T D^T K M W_{\hat{\tau}} D \hat{u}}. \tag{3.45}$$

Note that $\mathbf{W_u}$ represents the control volumes on the regular grid now.

Therefore, our algorithm can be divided into three steps. First, we interpolate the velocities on the regular grid and get the corresponding ones on the adaptive grid using (3.41). Then, we solve (3.44) with a linear equation solver such as incomplete Cholesky preconditioned conjugate gradient method (ICPCG). At last, we transfer the resultant velocities on the adaptive grid back to the regular grid with (3.45).

Chapter 4

Implementation and Results

We have discussed each step of a fluid simulation and placed emphasis on our novel viscosity solver, which is the main contribution of this thesis. In this chapter, we will present further implementation details, evaluate our method's accuracy, and provide a comparison against standard fluid simulation on a regular grid (e.g., the work in [Batty and Bridson, 2008]).

4.1 Implementation

Our simulation steps are outlined in Algorithm 1. Since it is trivial to advect marker particles and to integrate the force of gravity (both can be simulated with an ODE solver), we will not dedicate a section to them. Our C++ implementation uses Eigen [Guennebaud et al., 2010] to handle linear algebra operations, and solves the SPD systems during the pressure projection and the viscosity solve with the conjugate gradient method assisted by an incomplete Cholesky preconditioner. We create the quadtree data structure based on a stack of dense uniform grids. Each grid represents a level of the quadtree and has an affiliated bit array which marks the cells actually used in the data structure.

```
Algorithm 1: Viscous Fluid Simulation
Input :
        fluid density;
        MAC grid solid face areas;
        initial (divergence-free) MAC grid velocities;
        maximum time step size $\Delta t$;
        simulation stop time $t_f$.
Output:
        MAC level set value at each time step.

1  set $t = 0$
2  while $t < t_f$ do
3      use CFL condition to determine the actual time step $\delta t$ ;
4      advect marker particles ;
5      compute fluid's signed distance field (Section 2.4);
6      advect MAC grid velocities (Section 2.3);
7      apply gravity force to the MAC grid velocities;
8      apply viscosity force to the MAC grid velocities (Chapter 3);
9      perform pressure projection (Section 2.5);
10     extrapolate fluid velocities to zero-area faces;
11     increment $t$ by $\delta t$;
12 end
```

Algorithm 2 provides a detailed explanation of line 7 of Algorithm 1 and is the main contribution of this thesis. Our refinement strategy is to keep the liquid surface at the finest resolution, and aggressively coarsen the quadtree as we move further into the interior of the liquid. We therefore begin with the fine regular grid, and determine the first set of cells to coarsen by thresholding the distance field. As such, we merge leaf cells when

$$\phi(\vec{x}) > 1.5l,$$

where l is the length of a fine regular grid cell. Theoretically, we only need $\phi(\vec{x}) > l$ to ensure the free surface cells all locate at the leaf level of the quadtree. But since the level set field is just an approximation of the exact signed distances, we conservatively increase the threshold to $1.5l$ which we observed to work well under the various scenarios we tested. After creating this second to last level of the quadtree, we proceed to recursively merge cells whenever possible to do so while satisfying the constraint that the tree remains graded.

```
Algorithm 2: Apply Viscosity Force
  Input   :
            fluid density;
            fluid viscosity;
            time step size $\delta t$;
            regular grid velocities;
            fluid signed distance field;
            solid signed distance field.
  Output:
            updated regular grid velocities.

1 generate quadtree data structure based on the fluid signed distance field;
2 compute the linear interpolation operator mapping the velocities on the regular grid
    to the quadtree grid (Section 3.5);
3 compute the deformation rate operator on the quadtree grid (Section 3.4.1);
4 compute the control volumes (Sections 3.4.2 and 3.4.3);
5 build and solve the linear system (Section 3.3.2);
6 map the updated velocities on the quadtree grid to the regular grid (Section 3.5).
```

4.2 Comparison with Regular Grid Results

Since this algorithm is developed for fluid simulation in the context of computer graphics, we need to show it can generate similar visual behavior to its counterpart on a regular grid, but with fewer degrees of freedom. We use the variational finite difference method presented by [Batty and Bridson, 2008] as a baseline, and construct a quadtree grid based on the same regular grid. In other words, the size of the finest grid cells on the adaptive grid is the same as that of the regular grid cells. Figures 4.1 and 4.2 illustrate the results produced using the algorithm of Batty and Bridson [2008] and our own, respectively. The viscosity coefficient of the fluid is 1 kg/m·s, the maximum simulation step is 2 ms, and the domain size is 1m by 1m.

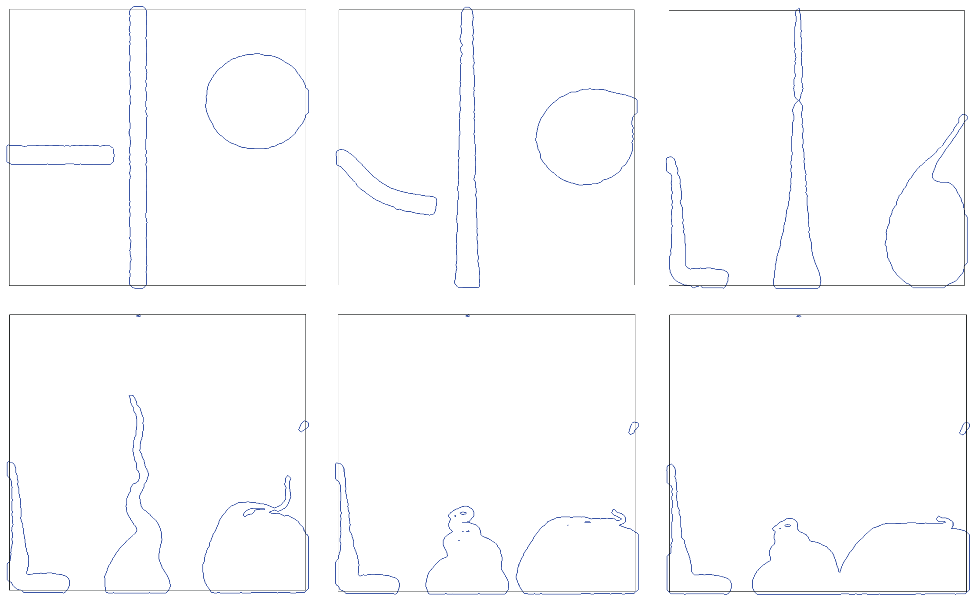

Figure 4.1: Screenshots of the fluid simulation on a regular grid at 0 ms, 80 ms, 160 ms, 240 ms, 320 ms and 400 ms.

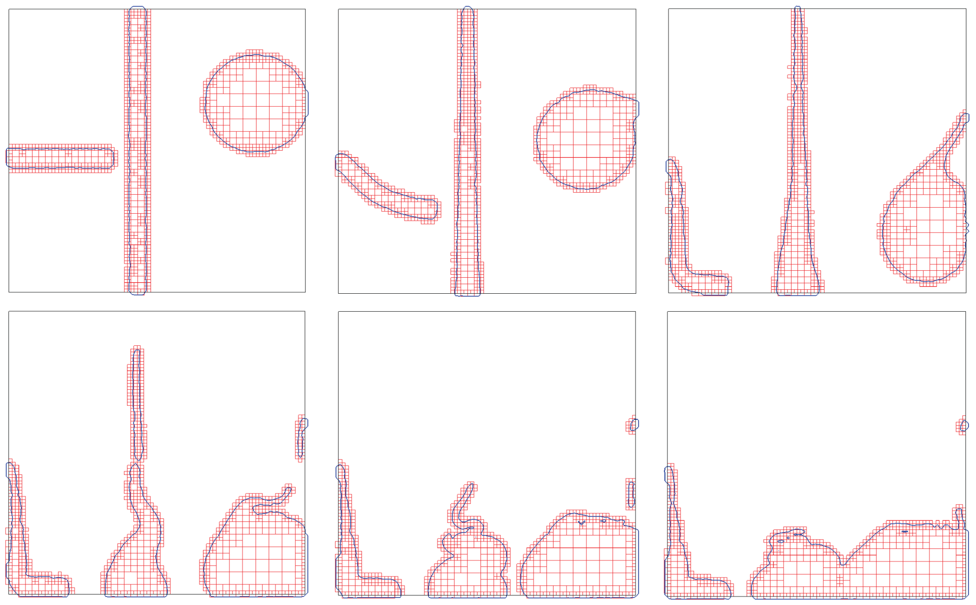

Figure 4.2: Screenshots of the fluid simulation on a graded quadtree grid at 0 ms, 80 ms, 160 ms, 240 ms, 320 ms and 400 ms. Coarser cells are used in the liquid interior.

Although the two animations are not exactly the same, our method preserves finely resolved details on the liquid's surface and yields buckling and folding behavior that is qualitatively consistent with that of the regular grid algorithm. Table 4.1 presents the average time of a complete simulation step, a viscosity integration step by itself, and a linear system solve within the viscosity integration step. We simulate 600 ms of fluid motion on a computer with 2.2 GHz Intel Core i7 CPUs and 16 GB 1600 MHz memory.

	No. of Time Steps	Total Step(ms)	Viscosity Step (ms)	Linear System (ms)	DOF
Regular grid	336	43.5	25.1	6.17	26311.0
Adaptive grid	358	82.6	63.3	3.46	13376.1

Table 4.1: Total number of time steps executed; average time spent on each simulation step, viscosity integration step, and linear system solve; the average number of velocity degrees-of-freedom in each case. The simulation runs from 0 to 600 ms.

The numbers of steps are different for the regular grid and the adaptive grid simulations because we set the time step size no larger than that dictated by the CFL condition and because the viscosity step in each case yields slightly different results, the two simulations diverge mildly over time. Our algorithm successfully decreases the DOFs by a half through the use of spatial adaptivity, and as such the corresponding linear system takes only about 60% as much time as its regular grid counterpart.

However, the viscosity simulation as a whole takes a longer time to finish on an adaptive grid. This is because we have focused on developing and evaluating the feasibility of an adaptive *numerical discretization* of viscosity, but our prototype implementation has not yet been highly optimized at the code level. Some operations, such as getting a leaf cell and retrieving a neighbouring node, may not be optimally efficient. In addition, since only the viscosity is solved on adaptive grids, we have to map the velocity fields between a regular grid and an adaptive grid before and after the simulation as explained in Section 3.5. This process takes around 20% of the total time to solve the viscosity step based on our tests. Therefore, there are two possible ways to reduce the simulation time. One is to build or use a more efficient quadtree data structure, such as a hierarchical sparse grid representation proposed by Setaluri et al. [2014]. The other is to get rid of the mapping process and simulate all steps on one adaptive grid by combining our method with the method of Losasso et al. [2006a].

We create a second example to further demonstrate the advantage of our algorithm, where more than half of the simulation domain is covered by the fluid. Such a scenario is ideal for our algorithm because it allows a much larger volume of liquid to be efficiently simulated at a coarser resolution. Figures 4.3 and 4.4 show the animations generated by the baseline algorithm and our own. The viscosity coefficient of the fluid is still set to 1 kg/m·s. Once again the results are qualitatively consistent.

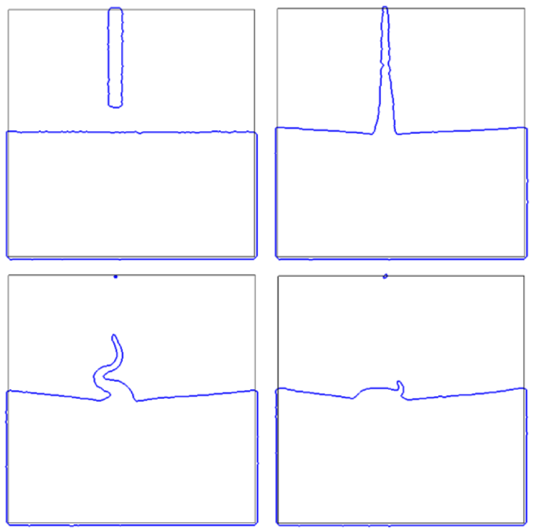

Figure 4.3: Screenshots of the fluid simulation on a regular grid at 0 ms, 100 ms, 200 ms, 300 ms.

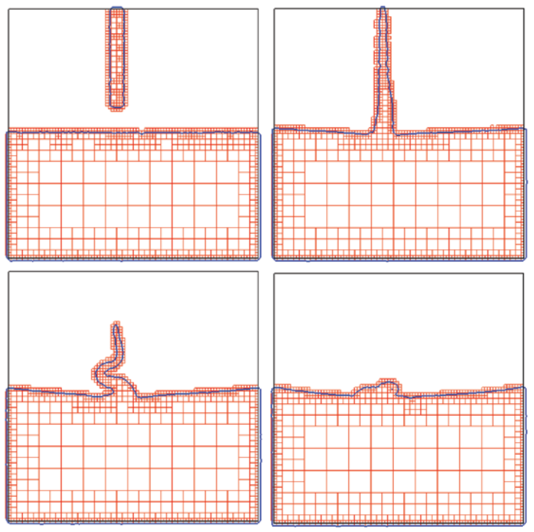

Figure 4.4: Screenshots of the fluid simulation on a graded quadtree grid at 0 ms, 100 ms, 200 ms, 300 ms. Different levels of coarse cells are used in the liquid interior.

Table 4.2 presents the average computation time of the two algorithms. Our method reduces the degrees of freedom by more than 80% and demonstrates the same simulation time as the regular grid method. Thus we can see that the advantages of the adaptive grid will depend significantly on the particular scene to be simulated. We are optimistic that by adopting the optimizations suggested above in future work, our approach will be able to surpass the speed of the baseline regular grid method.

	No. of Time Steps	Total Step (ms)	Viscosity Step (ms)	Linear System (ms)	DOF
Regular grid	301	89.2	56.1	20.1	76289.6
Adaptive grid	301	89.4	56.6	2.7	13505.5

Table 4.2: Total number of time steps executed; average time spent on each simulation step, viscosity integration step, and linear system solve; the average number of velocity degrees-of-freedom in each case. The simulation runs from 0 to 600 ms.

4.3 Rate of Convergence

In this section, we will present the results of our numerical experiments intended to evaluate the convergence rate of our viscosity solver on quadtree grids. While the primary intended application of our method is computer animation, where visuals may at times take priority over strict numerical accuracy, it is nevertheless important to validate that our method indeed correctly solves the PDE under consideration. To focus our analysis on the effects of our proposed quadtree discretization (rather than the influence of irregular free surface boundaries) and to allow for the straightforward derivation of analytical reference solutions, we test the solver on several prescribed quadtree grids with different initial conditions on closed square-shaped domains. Note that our numerical tests focus solely on our novel viscosity step alone, and do not involve the remaining components of the regular grid fluid solver.

Example 1: To derive our first test case, we first define the container where the simulation is conducted as a $\pi \times \pi$ square box, and set the velocity field at $t = \Delta t$ to

$$\begin{aligned} u(x, y, t = \Delta t) &= \sin(x)\sin(y); \\ v(x, y, t = \Delta t) &= \sin(x)\sin(y). \end{aligned} \tag{4.1}$$

Equation 4.1 satisfies a Dirichlet boundary condition that $u = v = 0$ on the boundary, Γ_f. Since only the accuracy versus the grid cell size is investigated, we can calculate the velocity at $t = 0$ with the semi-discrete form of the viscosity equation (3.8) by analytically integrating backwards in time by one step of length Δt, to arrive at:

$$\begin{aligned} u(x,y,t=0) &= \left(1+\frac{3\mu\Delta t}{\rho}\right)\sin(x)\sin(y) - \frac{\mu\Delta t}{\rho}\cos(x)\cos(y) \\ v(x,y,t=0) &= \left(1+\frac{3\mu\Delta t}{\rho}\right)\sin(x)\sin(y) - \frac{\mu\Delta t}{\rho}\cos(x)\cos(y). \end{aligned} \tag{4.2}$$

Then, we can use (4.2) as the input to our viscosity solver step, and compare its output against the exact solution provided by (4.1). We measure the absolute value of the error at each face position, and take the largest of these to find the L_∞ error for a given grid resolution. We test our algorithm on the following two quadtree grids.

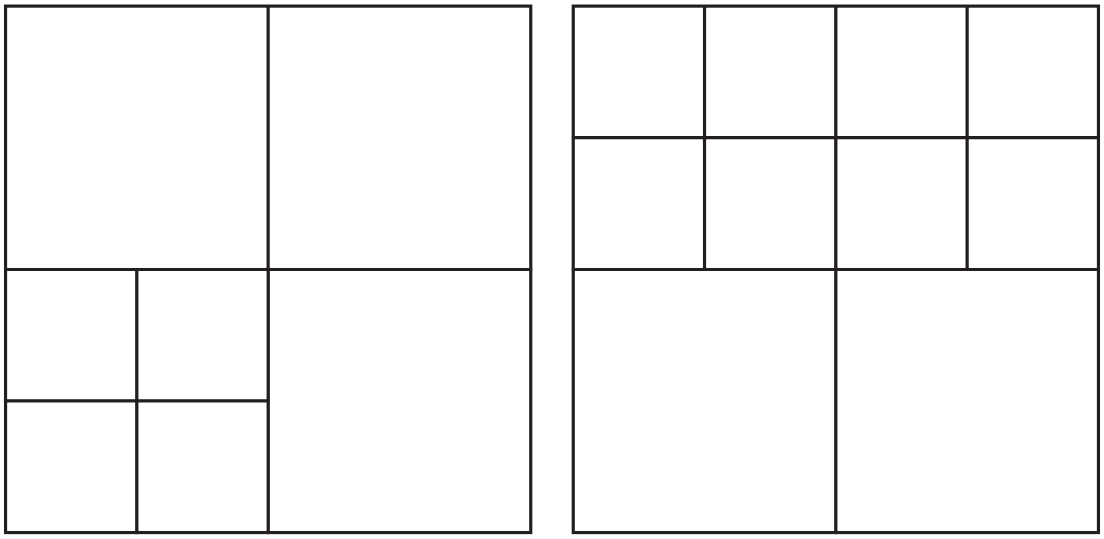

Figure 4.5: Left: only a quarter of the grid is refined; Right: top half of the grid is refined.

The grids illustrated by Figure 4.5 are the base-level (coarsest) grids in the test, and we evenly divide each grid cell into 4 sub-squares to generate the grid on each subsequent level. If our method is first order accurate, we would expect that as we continually refine the grid, the error observed at each refinement level will be approximately half that of the preceding level. We set Δt, ρ and μ to 1 in this example for simplicity. Tables 4.3 and 4.4 show the convergence results on the corner-refined and the top-half-refined grids. Since the corner-refined grid is symmetric with respect to the diagonal line, the errors for u and v are identical which explains why Table 4.3 only displays the results for u.

Cells	$\|\tilde{u} - u\|_\infty$	order
7	1.23×10^{-1}	-
7×4^1	5.86×10^{-2}	1.07
7×4^2	3.17×10^{-2}	0.89
7×4^3	1.64×10^{-2}	0.95
7×4^4	8.33×10^{-3}	0.98
7×4^5	4.19×10^{-3}	0.99
7×4^6	2.10×10^{-3}	1.00

Table 4.3: L_∞ error of velocities in viscosity solve on a corner refined quadtree grid.

Cells	$\|\tilde{u} - u\|_\infty$	order	$\|\tilde{v} - v\|_\infty$	order
10	1.38×10^{-1}	-	1.42×10^{-1}	-
10×4^1	5.05×10^{-2}	1.45	6.49×10^{-2}	1.13
10×4^2	1.55×10^{-2}	1.70	3.25×10^{-2}	1.00
10×4^3	4.24×10^{-3}	1.87	1.65×10^{-2}	0.98
10×4^4	1.62×10^{-3}	1.39	8.33×10^{-3}	0.99
10×4^5	8.12×10^{-4}	1.00	4.19×10^{-3}	0.99
10×4^6	4.06×10^{-4}	1.00	2.10×10^{-3}	1.00

Table 4.4: L_∞ error of velocities in viscosity solve on a top half refined quadtree grid.

To see the convergence behavior visually, we can draw a convergence graph in Figure 4.6 with the results in Table 4.3 and Table 4.4. On this loglog plot, we can see that under refinement the slope of our error plot closely matches that of a first order method.

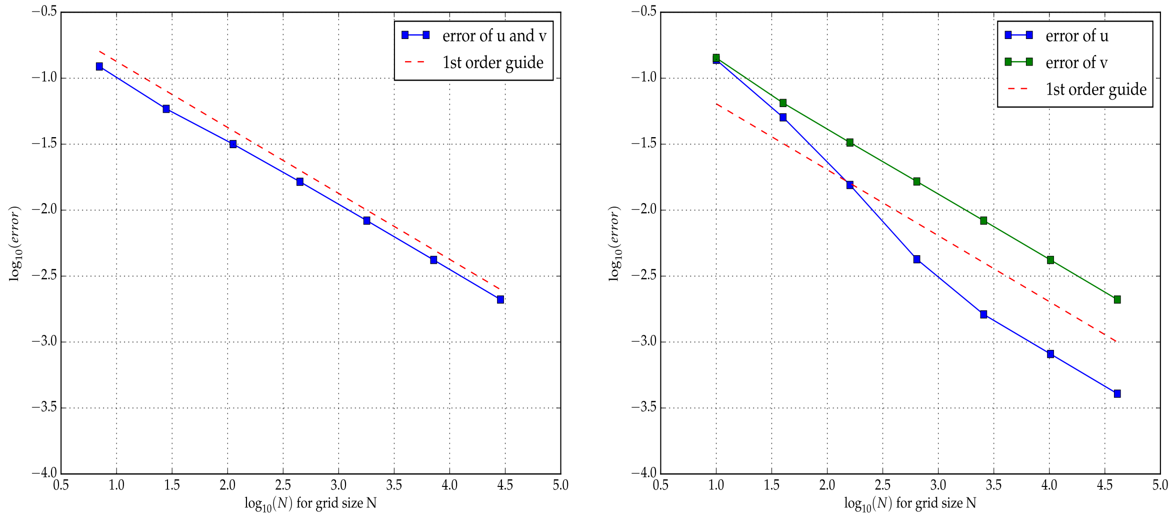

Figure 4.6: Left: error of u or v on the corner-refined grid; Right: error of u and v on the top-half-refined grid.

Example 2: For a slightly more complex case, we set u and v to different functions. At $t = \Delta t$,

$$\begin{aligned} u(x, y, t = \Delta t) &= \sin(x)\sin(y), \\ v(x, y, t = \Delta t) &= (x^2 - \pi x)(y^2 - \pi y). \end{aligned} \tag{4.3}$$

Using the same approach to derive the corresponding analytical input function as in the previous example, we can get u and v at $t = 0$:

$$\begin{aligned} u(x, y, t = 0) &= \sin(x)\sin(y) - \frac{\mu \Delta t}{\rho}\left[(2x - \pi)(2y - \pi) - 3\sin(x)\sin(y)\right], \\ v(x, y, t = 0) &= (x^2 - \pi x)(y^2 - \pi y) - \frac{\mu \Delta t}{\rho}\left[\cos(x)\cos(y) + 4(x^2 - \pi x) + 2(y^2 - \pi y)\right]. \end{aligned} \tag{4.4}$$

The grids we used in Example 1 only contain two levels of cells. In order to demonstrate that our algorithm works well on (graded) multi-level grids, we test it on the following grid.

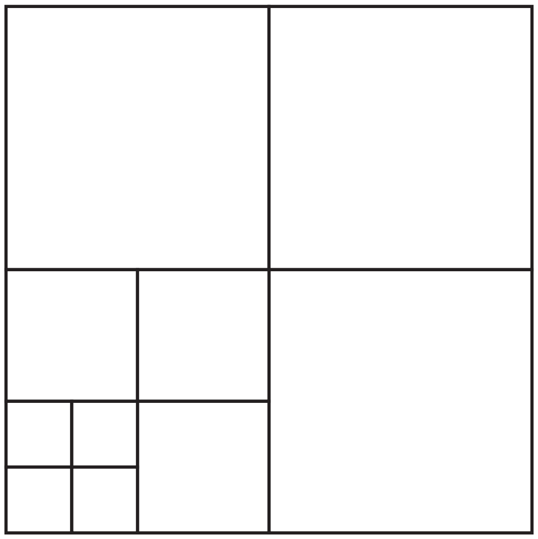

Figure 4.7: Corner-refined grid containing three levels of cells.

We still set Δt, ρ and μ to 1 in this example and present the test results in Table 4.5. Although the grid is still symmetric, our analytical u and v have different forms so the convergence results are not identical. In any case, we once again observe first order convergence behavior. Figure 4.8 shows the corresponding convergence graph.

Cells	$\|\tilde{u} - u\|_\infty$	order	$\|\tilde{v} - v\|_\infty$	order
10	1.64×10^{-1}	-	9.54×10^{-1}	-
10×4^1	9.17×10^{-2}	0.84	5.12×10^{-1}	0.90
10×4^2	4.77×10^{-2}	0.94	2.59×10^{-1}	0.98
10×4^3	2.40×10^{-2}	0.99	1.30×10^{-1}	0.99
10×4^4	1.19×10^{-2}	1.01	6.51×10^{-2}	1.00
10×4^5	5.93×10^{-3}	1.00	3.26×10^{-2}	1.00
10×4^6	3.04×10^{-3}	0.96	1.63×10^{-2}	1.00

Table 4.5: L_∞ error of velocities in viscosity solve on a three-level corner refined quadtree grid.

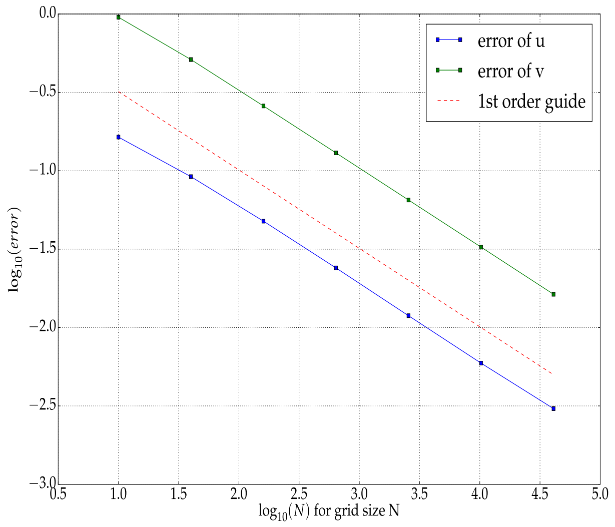

Figure 4.8: Error of u and v on the three-level corner refined grid.

Example 3: As we mentioned in Section 3.2.1, an advantage of using the full form of the viscosity equation over the Laplacian form is that it can simulate a fluid with spatially varying viscosity. In this example, we want to demonstrate that fact on adaptive grids. Suppose that the viscosity only changes along the x-axis and has the form:

$$\mu(x) = \frac{x}{\pi} + 0.5.$$

The analytical solution we set at $t = \Delta t$ is exactly the same as that in Example 1, which is

$$\begin{aligned} u(x, y, t = \Delta t) &= \sin(x)\sin(y), \\ v(x, y, t = \Delta t) &= \sin(x)\sin(y). \end{aligned} \tag{4.5}$$

However, since the viscosity is no longer constant, the values of u and v at $t = 0$ will be different. We calculate the corresponding input functions via the semi-discrete equation again.

$$
\begin{aligned}
u(x,y,t=0) =& \sin(x)\sin(y) \\
&- \frac{\Delta t}{\rho}\left[\frac{2}{\pi}\cos(x)\sin(y) + (\cos(x+y) - 2\sin(x)\sin(y))\left(\frac{x}{\pi}+0.5\right)\right], \\
v(x,y,t=0) =& \sin(x)\sin(y) \\
&- \frac{\Delta t}{\rho}\left[(\cos(x)\cos(y) - 3\sin(x)\sin(y))\left(\frac{x}{\pi}+0.5\right) + \frac{1}{\pi}\sin(x+y)\right].
\end{aligned}
\tag{4.6}
$$

Tables 4.6, 4.7 and 4.8 show the results on all three of the previously used base-level grids, which again indicate first order accuracy.

Cells	$\|\tilde{u}-u\|_\infty$	order	$\|\tilde{v}-v\|_\infty$	order
7	9.82×10^{-2}	-	1.22×10^{-1}	-
7×4^1	6.13×10^{-2}	0.68	5.17×10^{-2}	1.24
7×4^2	3.25×10^{-2}	0.92	3.12×10^{-2}	0.73
7×4^3	1.66×10^{-2}	0.97	1.66×10^{-2}	0.91
7×4^4	8.39×10^{-3}	0.98	8.45×10^{-3}	0.97
7×4^5	4.21×10^{-3}	0.99	4.24×10^{-3}	0.99
7×4^6	2.11×10^{-3}	1.00	2.12×10^{-3}	1.00

Table 4.6: L_∞ error of velocities in viscosity solve on a corner refined quadtree grid.

Cells	$\|\tilde{u}-u\|_\infty$	order	$\|\tilde{v}-v\|_\infty$	order
10	1.35×10^{-1}	-	1.53×10^{-1}	-
10×4^1	4.64×10^{-2}	1.54	6.26×10^{-2}	1.29
10×4^2	1.45×10^{-2}	1.68	3.24×10^{-2}	0.95
10×4^3	3.87×10^{-3}	1.91	1.67×10^{-2}	0.96
10×4^4	1.66×10^{-3}	1.22	8.45×10^{-3}	0.98
10×4^5	8.20×10^{-4}	1.02	4.29×10^{-3}	0.98
10×4^6	4.08×10^{-4}	1.01	2.17×10^{-3}	0.98

Table 4.7: L_∞ error of velocities in viscosity solve on a top half refined quadtree grid.

The convergence graphs on different grids are given as follows. The matching slopes illustrate that our algorithm can address problems involving varying viscosity without any issue and still produce first order accurate results.

Cells	$\|\tilde{u}-u\|_\infty$	order	$\|\tilde{v}-v\|_\infty$	order
10	1.25×10^{-1}	-	1.22×10^{-1}	-
10×4^1	6.33×10^{-2}	0.98	5.60×10^{-2}	1.12
10×4^2	3.24×10^{-2}	0.97	3.12×10^{-2}	0.84
10×4^3	1.66×10^{-2}	0.96	1.66×10^{-2}	0.91
10×4^4	8.37×10^{-3}	0.99	8.44×10^{-3}	0.98
10×4^5	4.20×10^{-3}	0.99	4.24×10^{-3}	0.99
10×4^6	2.11×10^{-3}	0.99	2.12×10^{-3}	1.00

Table 4.8: L_∞ error of velocities in viscosity solve on a three level corner refined quadtree grid.

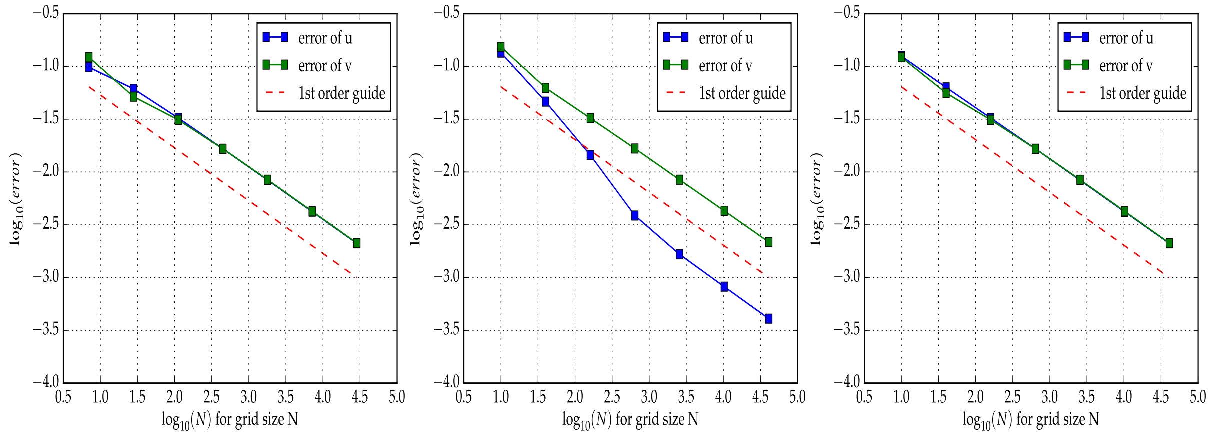

Figure 4.9: Left: Error of u and v on the corner refined grid; Middle: Error of u and v on the top refined grid; Right: Error of u and v on the three-level corner refined grid.

Chapter 5

Conclusion

In this thesis, we have thoroughly reviewed the research on viscous fluid simulation in computer graphics, and derived a symmetric positive-definite linear system from the variational form of the viscosity equation to solve viscosity on an adaptive grid. Since our method only refines the grid on the liquid interior and adopts the method of [Batty and Bridson, 2008] to treat the surface, it can naturally handle the complex free surface boundary condition of viscous fluids. We have also demonstrated that our algorithm is able to generate first-order accurate results and produce similar visual details compared to the regular grid algorithm, while using far fewer degrees-of-freedom in the simulation. In addition, a novel mapping method has been introduced to transfer the velocity fields between an adaptive grid and a regular grid, which can be applied to embed our adaptive viscosity solver into any standard Eulerian or FLIP/PIC inviscid fluid simulator on a regular grid.

5.1 Future Work

We have provided the mathematical expressions and a 2D implementation in this thesis, so an intuitive extension would be a 3D implementation which is more useful in the context of computer graphics. Since a 3D simulation normally involves more degrees-of-freedom than a 2D one for a given grid resolution, we expect that our algorithm would further reduce computation time spent in solving a linear system and thereby increase the simulation speed. Additionally, we can combine our algorithm with the method of Losasso et al. [2006a] that solves pressure and all the other fluid simulation steps on an adaptive grid. This will entirely avoid the need to perform mappings between grids. As a matter of fact, we have designed our method so that we don't need to modify our existing tree structure to accommodate this change since Losasso et al. [2006a] stores velocity samples at the same positions as ours. Another interesting research topic to investigate is how to solve the pressure and viscosity together to achieve viscous coiling in 3D as explained in [Larionov et al., 2017], but on an adaptive grid. Since the current method of [Larionov et al., 2017] is a few times slower than the operator splitting method of [Batty and Bridson, 2008], the adoption of adaptive grids can hopefully minimize the difference. As mentioned in Section 4.2, a better designed tree data structure and associated algorithms could be utilized to accelerate the simulation speed and reduce memory usage, such as the SPGrid [Setaluri et al., 2014]. Setaluri et al. [2014]'s work considered not only improving the pointer-based representation of the tree, but also the development of an optimized parallel implementation. Finally, our algorithm assumes that the adaptive grid it works on is always graded which somewhat simplifies the implementation. We would like to explore if it is possible to produce convergent results on a non-graded tree while ensuring the system remains symmetric positive definite at the same time.

References

Aanjaneya, M., Gao, M., Liu, H., Batty, C., and Sifakis, E. (2017). Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics (TOG)*, 36(4):140.

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 (TOG)*, 26(3):16.

Batchelor, G. K. (1967). *An Introduction to Fluid Dynamics*. Cambridge University Press.

Bathe, K.-J. and Wilson, E. L. (1976). *Numerical Methods in Finite Element Analysis*, volume 197. Prentice-Hall Englewood Cliffs, NJ.

Batty, C. (2017). A cell-centred finite volume method for the poisson problem on non-graded quadtrees with second order accurate gradients. *Journal of Computational Physics*, 331:49–72.

Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 26(3):100.

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. Eurographics Association.

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. ACM.

Blinn, J. F. (1982). A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256.

Brackbill, J. and Ruppel, H. (1986). Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2):314–343.

Bridson, R. (2015). *Fluid Simulation for Computer Graphics.* CRC Press.

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. ACM.

Coirier, W. J. (1994). *An Adaptively-refined, Cartesian Cell-based Scheme for the Euler and Navier-Stokes Equations*, volume 106754. University of Michigan PhD thesis.

Enright, D., Fedkiw, R., Ferziger, J., and Mitchell, I. (2002a). A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116.

Enright, D., Marschner, S., and Fedkiw, R. (2002b). Animation and rendering of complex water surfaces. *ACM Transactions on Graphics (TOG)*, 21(3):736–744.

Fält, H. and Roble, D. (2003). Fluids with extreme viscosity. In *ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1. ACM.

Fedkiw, R. and Osher, S. (2003). *Level Set Methods and Dynamic Implicit Surfaces.* Springer Verlag New York.

Fedkiw, R., Stam, J., and Jensen, H. W. (2001). Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22. ACM.

Fefferman, C. L. (2006). Existence and smoothness of the Navier-Stokes equation. *The Millennium Prize Problems*, pages 57–67.

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

Foster, N. and Fedkiw, R. (2001). Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 23–30. ACM.

Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483.

Gerya, T., May, D., and Duretz, T. (2013). An adaptive staggered grid finite difference method for modeling geodynamic stokes flows with strongly variable viscosity. *Geochemistry, Geophysics, Geosystems*, 14(4):1200–1225.

Ghia, U., Ghia, K. N., and Shin, C. (1982). High-re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411.

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.

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

Guennebaud, G., Jacob, B., et al. (2010). Eigen. *URl: http://eigen. tuxfamily. org.*

Guittet, A., Theillard, M., and Gibou, F. (2015). A stable projection method for the incompressible Navier-Stokes equations on arbitrary geometries and adaptive quad/octrees. *Journal of Computational Physics*, 292:215–238.

Harlow, F. H. (1962). The particle-in-cell method for numerical solution of problems in fluid dynamics. Technical report, Los Alamos Scientific Lab., N. Mex.

Harlow, F. H. and Welch, J. E. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189.

Hong, J.-M. and Kim, C.-H. (2005). Discontinuous fluids. *ACM Transactions on Graphics (TOG)*, 24(3):915–920.

Houston, B., Bond, C., and Wiebe, M. (2003). A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1. ACM.

Khokhlov, A. M. (1998). Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, 143(2):519–543.

Kim, T. and Lin, M. C. (2007). Fast animation of lightning using an adaptive mesh. *IEEE Transactions on Visualization and Computer Graphics*, 13(2).

Larionov, E., Batty, C., and Bridson, R. (2017). Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Transactions on Graphics (TOG).*

Limache, A., Idelsohn, S., Rossi, R., and Oñate, E. (2007). The violation of objectivity in laplace formulations of the Navier–Stokes equations. *International Journal for Numerical Methods in Fluids*, 54(6-8):639–664.

Lipnikov, K., Manzini, G., and Shashkov, M. (2014). Mimetic finite difference method. *Journal of Computational Physics*, 257:1163–1227.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 163–169. ACM.

Losasso, F., Fedkiw, R., and Osher, S. (2006a). Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids*, 35(10):995–1010.

Losasso, F., Gibou, F., and Fedkiw, R. (2004). Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (TOG)*, 23(3):457–462.

Losasso, F., Shinar, T., Selle, A., and Fedkiw, R. (2006b). Multiple interacting liquids. *ACM Transactions on Graphics (TOG)*, 25(3):812–819.

Miller, G. and Pearce, A. (1989). Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305–309.

Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574.

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. Eurographics Association.

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.

Nielsen, M. B. and Bridson, R. (2016). Spatially adaptive flip fluid simulations in bifrost. In *ACM SIGGRAPH 2016 Talks*, page 41. ACM.

Perot, J. B. and Subramanian, V. (2007). Discrete calculus methods for diffusion. *Journal of Computational Physics*, 224(1):59–81.

Platt, D. T. J. and Fleischer, K. (1989). Heating and melting deformable models (from goop to glop). In *Proc. Graphics Interface*, volume 89, pages 219–226.

Popinet, S. (2003). Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572–600.

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. Eurographics Association.

Roble, D., Zafar, N. b., and Falt, H. (2005). Cartesian grid fluid simulation with irregular boundary voxels. In *ACM SIGGRAPH 2005 Sketches*, page 138. ACM.

Schroeder, C., Zheng, W., and Fedkiw, R. (2012). Semi-implicit surface tension formulation with a Lagrangian surface mesh on an eulerian simulation grid. *Journal of Computational Physics*, 231(4):2092–2115.

Setaluri, R., Aanjaneya, M., Bauer, S., and Sifakis, E. (2014). Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)*, 33(6):205.

SideFX (2017). Houdini.

Stam, J. (1999). Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co.

Staniforth, A. and Côté, J. (1991). Semi-Lagrangian integration schemes for atmospheric modelsa review. *Monthly Weather Review*, 119(9):2206–2223.

Sussman, M., Almgren, A. S., Bell, J. B., Colella, P., Howell, L. H., and Welcome, M. L. (1999). An adaptive level set approach for incompressible two-phase flows. *Journal of Computational Physics*, 148(1):81–124.

Taylor, C. and Hood, P. (1973). A numerical solution of the Navier-Stokes equations using the finite element technique. *Computers & Fluids*, 1(1):73–100.

Tsai, Y.-h. R. (2002). Rapid and accurate computation of the distance function using grids. *Journal of Computational Physics*, 178(1):175–195.

Zhang, X., Bridson, R., and Greif, C. (2015). Restoring the missing vorticity in advection-projection fluid solvers. *ACM Transactions on Graphics (TOG)*, 34(4):52.

Zhu, Y. and Bridson, R. (2005). Animating sand as a fluid. *ACM Transactions on Graphics (TOG)*, 24(3):965–972.