The Mimetic Approach to Incompressible Surface Tension Flows

by

Egor Larionov

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, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Water has many aesthetic properties that can have a strong impact on our perceptions. For instance, coffee can bring a feeling of liveliness, rain drops on a window may spark nostalgia, morning dew on a leaf can suggest freshness, and melting icicles remind us of spring. The ubiquity of water and the complexity of phenomena it can exhibit makes it an important and interesting topic for simulation in computer graphics, where the focus lies in visual aesthetics.

Our focus is to produce a method for simulating water droplets at scales where surface tension effects are visually significant. These are precisely the scales in the scenarios mentioned above. We propose a simple new approach to simulating water-like liquids with visible surface tension effects in two dimensions. We employ a recently developed Mimetic Finite Difference (MFD) method to solve the Poisson problem, which enforces incompressibility in the incompressible Euler equations. Using a cut-cell discretization, the MFD method allows us to extend the ubiquitous finite difference method on a Marker-and-Cell (MAC) discretization to handle irregular boundaries. To produce surface tension effects, we keep track of an explicit Lagrangian surface that conforms exactly to the simulation mesh. To achieve stable results, we adapt a semi-implicit surface tension scheme [Misztal et al., 2012] to our MFD pressure solve, which allows us to use time steps about 2-3 times larger than the corresponding explicit method. In addition, the semi-implicit method is extended to simulate liquids in contact with hydrophobic and hydrophilic surfaces. To provide stable surface tracking, we employ a method based on marching square templates [Rocchini et al., 2001; Müller, 2009] augmented by two simple techniques for improving mesh quality. Collapsing small interior grid edges near the fluid surface eliminates triangle elements with small angles near the liquid surface. Eliminating cells with small angles gives a better bound on the conditioning of the discrete Laplacian used in the Poisson solve, hence adding stability to our simulation. To compute surface tension forces, we use a version of the surface mesh that has been perturbed to reduce the number of short edges, in order to more robustly estimate surface curvature. These are essential for stability when coupling the MFD solve with surface tension forces. Our approach employs a unique combination of methods to address the problem of accurately tracking the contact between liquid and solid surfaces. We propose a method that couples well with the majority of fluid simulators used in the visual effects industry, while introducing a stable surface tension technique that doesn't require complex auxiliary meshing strategies [Zheng et al., 2015].

Acknowledgements

First and foremost I would like to thank my supervisor Christopher Batty for providing the much needed guidance and mentorship throughout my studies on a regular basis. His enthusiasm, intelligence and intuition has constantly been a driving force that kept me on my toes, eager to learn and tackle challenging problems.

I am grateful to Stephen Mann for initially accepting me as his Master's student, and giving me the opportunity to pursue full-time research in computer graphics. Under his supervision I was able to experience a broad spectrum of topics and discover my true passions.

I would like to thank my committee members, Gladimir Baranoski and Justin Wan for your thorough analysis of this project. Your challenging questions and insightful comments gave me a fresh perspective on my work and improved the contents of this thesis.

Starting from my internship at Side Effects Software (SideFX) in Toronto, I was able to experience research related work in the computer graphics industry thanks to my first supervisor, Stephen Mann and the director of product development at SideFX, Cristin Barghiel who hired me. My supervisor at SideFX, Michiel Hagedoorn, gave me the opportunity to learn a tremendous amount about cutting edge meshing techniques, robust floating point arithmetic and rigorous and effective C++ programming. His passion for mathematics and rigorous approach to programming has significantly influenced my work. Following this internship, Cristin also provided me with a chance to work part time at SideFX under Michiel's supervision for the rest of my time as a Master's student.

My colleagues at the Computational Motion Group (CMG) including Ryan, Dustin, Vinicius, Yipeng, Filipe, and Omar, have showed me the true value in collaborative research, and our discussions taught me a great deal of what I know about physically based animation today. My friends and lab mates Eddie, Parsiad, David, Colleen, Philippe, Spencer, Kai, Nick, Luyu, Shan, including everybody from CMG and the scientific computing laboratory have made my research experience enjoyable and truly memorable. I will surely miss all the astute philosophical discussions we've had, mathematical problems we've tackled for exercise, and innovative startup ideas we've shared. I would also like to thank everyone at the Computer Graphics Laboratory (CGL) for giving me a broad perspective on computer graphics research.

I am deeply grateful to my parents Serguei and Valentina. This work wouldn't be possible without your unconditional love, support and understanding. Thank you also for your impartial advice and guidance, helping me choose a meaningful career path.

I am indebted to oldest friends Dan and Mihai for keeping me level-headed and honest all these years. Finally, I would like to thank Dian Xiang for making the months of non-stop writing and programming bearable, and showing me the value in work-life balance.

Thank you also to those whose names I missed, but rest assured that I remember all the faces that made an impact on my work.

Dedication

To my parents, Serguei and Valentina, and my brothers, Ilia and Boris.

Table of Contents

List of Tables

List of Figures

Nomenclature

The following are commonly occurring symbols used throughout this thesis. Units are omitted in dimensionless quantities.

Δt	Time step, s
Δx	Size of one grid cell (width and/or height), m
ν	Kinematic viscosity coefficient, $\mathrm{m}^2\,\mathrm{s}^{-1}$
ρ	Density, $\mathrm{kg}\,\mathrm{m}^{-3}$
$\vec{f}$	Acceleration due to external forces, $\mathrm{m}\,\mathrm{s}^{-2}$
$\vec{f}_1, \vec{f}_2$	Constituents of the external acceleration $\vec{f} = \vec{f}_1 + \vec{f}_2$, $\mathrm{m}\,\mathrm{s}^{-2}$
$\vec{f}_{st}$	Force of surface tension
$\vec{g}$	Gravitational acceleration, $\mathrm{m}\,\mathrm{s}^{-2}$
$\vec{u}$	Velocity, $\mathrm{m}\,\mathrm{s}^{-1}$
$\vec{u}^*$	Intermediate velocity after advection, $\mathrm{m}\,\mathrm{s}^{-1}$
$\vec{u}^{**}$	Intermediate velocity after advection and body forces, $\mathrm{m}\,\mathrm{s}^{-1}$
p	Pressure, Pa
t	Time, s
$\frac{D\vec{u}}{Dt}$	Total derivative of velocity with respect to time
∇	Continuous gradient operator

$\nabla\cdot$	Continuous divergence operator
Γ^D	Part of the liquid boundary subject to Dirichlet boundary condition
Γ^N	Part of the liquid boundary subject to Neumann boundary condition
$\hat{n}$	Outward normal unit vector
Ω	Liquid domain
$\partial\Omega$	Liquid domain boundary
f^D	Function defining the pressure on the liquid-air boundary for the Dirichlet boundary condition
f^N	Function defining the velocity normal to the liquid-solid boundary for the Neumann boundary condition
e, c	Symbols used to represent mesh edges and cells respectively
n, m	Symbols used to represent distinct mesh nodes
N_C	Number of cells on the discrete mesh
N_E	Number of edges on the discrete mesh
N_V	Number of nodes on the discrete mesh
N_{SV}	Number of surface vertices, which correspond to nodes on the surface of the discrete mesh
$\mathbf{p}$	Stacked vector in $\mathbb{R}^{N_V}$ of pressure samples
$\mathbf{u}$	Stacked vector of velocity samples in $\mathbb{R}^{N_E}$ tangential to mesh edges
$\mathbf{u}^{**}$	Stacked vector of intermediate velocity samples in $\mathbb{R}^{N_E}$ discretizing the continuous velocity field $\vec{u}^{**}$
λ	Scaled pressure acting as a Lagrange multiplier, equivalent to $\Delta t\mathbf{p}$
$\mathbf{D}$	Discrete divergence operator
$\mathbf{G}$	Discrete gradient operator

$\mathbf{M}_E, \mathbf{M}$ Edge-based discrete inner product matrix on $\mathbb{R}^{N_E}$

$\mathbf{M}_V$ Node-based discrete inner product matrix on $\mathbb{R}^{N_V}$

$\tilde{\mathbf{D}}$ Simplified discrete divergence operator synonymous to $\mathbf{G}^\top \mathbf{M}_E$

Chapter 1

Introduction

Water is an essential part of life on Earth. It is all around us; we interact with it every day, from the coffee we drink in the morning, to the warm bath we take to comfort us from the daily grind. It is also found as a main component in most liquid substances we see on a daily basis. For example, blood, sweat, milk, juice, are all mainly composed of water. It is no wonder why simulating water-like liquids is such an important topic in computer graphics. Mathematically, common fluids are typically described by the Navier-Stokes equations. However, water-like liquids are typically highly incompressible and possess relatively low viscosity and constant density. This allows for a simpler model using the incompressible Euler equations instead. This model has been thoroughly studied and widely used in the field of computational fluid dynamics for scientific and engineering purposes as well as computer graphics for physically-based animation. In science and engineering, fidelity of internal fluid quantities (e.g. temperature and pressure) is vitally important and fluids are typically studied in closed systems. In computer graphics, however, the focus is on the aesthetics of the fluid and how it looks on the surface. As a result more computational focus is put towards the dynamics on the surface than the internal mechanics of a fluid. Therefore, the methods employed in science and engineering are not always suitable for computer graphics purposes. This motivates the research of new simulation methods with a focus on visual aesthetics.

One visually interesting property of water is surface tension, which is especially visible at small scales, like on water droplets. This effect is particularly important because it characterizes the shape of a droplet resting on a solid surface. We can observe water droplets in a variety of settings like morning dew on leaves, condensation on a cool bottle, or rain droplets on a window.

Consider a water droplet floating in the air. In this scenario, surface tension is an inward force per unit area on the liquid surface. This force has the effect of minimizing the surface area of the droplet. More precisely, it is caused by the imbalance of *adhesive* forces between water and air molecules, and *cohesive* forces between water molecules alone. Specifically, cohesive forces overpower adhesive forces, causing the interior water molecules to pull the surface inwards as in Figure 1.1. Of course, it is uncommon to see a droplet suspended in the air for a long period of time. Due to gravity, the liquid water we typically see, rests on a solid surface. In this setting, the force of surface tension can cause the liquid to spread along a solid boundary, or repel the liquid away, characterizing "hydrophilic" and "hydrophobic" materials respectively. This phenomenon can only be seen where the boundaries of three or more materials intersect, forming a *contact curve*[1]. In our case, the materials are water, solid and air.

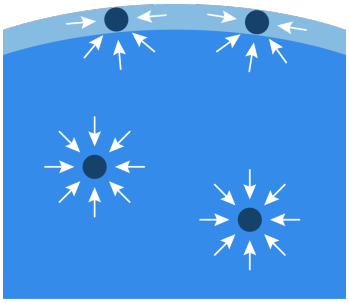


Figure 1.1: Imbalance of attractive forces on a liquid surface.

Our work focuses on the core of simulating liquids to allow for a straightforward inclusion of surface tension effects and an accurate boundary representation, especially near the contact curve. We aim to develop a framework that extends the dominant family of grid based simulation schemes in computer graphics, to allow existing implementations to benefit directly from our contributions.

1.1 Related Work

The majority of the methods for simulating liquids can be categorized as Lagrangian, where the discretization moves with the liquid, or Eulerian, where the discretization is stationary with respect to a moving liquid. The two most notable Lagrangian methods can be further split into particle methods such as Smoothed Particle Hydrodynamics (SPH), e.g. [Ihmsen et al., 2014b] where the fluid equations are solved directly on disconnected particles, and mesh methods using the finite element (FE) method or the like, e.g. [Clausen et al., 2013]. However, the majority of modern fluid simulators used in the computer graphics industry (e.g., [Autodesk, 2015; SideFX, 2015]) use Eulerian grid-based methods, or hybrids between Eulerian and Lagrangian approaches, in favour of purely Lagrangian methods. This is the main reason we chose to extend the Eulerian approach in our work. Eulerian schemes are

[1] In two dimensions we call this a *contact point*.

particularly simple, easy to implement, and efficient; they can even be used to improve existing SPH simulators [Cornelis et al., 2014]. At the core of a standard Eulerian grid-based method lies a regular grid (or sometimes an octree) that stores pressure values at cell centers and velocity components between pressure values (see Figure 3.3b for an example of the staggered marker-and-cell (MAC) discretization [Harlow et al., 1965]). This discretization is used to solve for incompressibility in the Euler equations using a finite difference (FD) scheme, and is often referred to as the pressure projection step. Other Eulerian methods typically involve the finite volume (FV) method, which generalizes the MAC grid with FD to more complex meshes with certain orthogonality constraints. Unfortunately, the accuracy of the basic FV method is restricted to cells with faces orthogonal to the line connecting the face and cell centroids. Since our discretization includes cells that violate this orthogonality constraint we seek an alternative method that ultimately reduce to the MAC grid with FD on the interior of the mesh while supporting more general cell geometries near boundaries. More background on our choice of method to solve for pressure projection is found in Chapter 3.

In addition to solving for incompressibility, a fluid solver also needs a means to advect[2] the velocity field as well as the fluid geometry itself, which can be in the form of particles, a mesh or an implicit function sampled on a grid. Among Eulerian methods, there are a variety of approaches for velocity advection, but the most dominant in industry are Lagrangian particle advection schemes. The particle-in-cell (PIC) method [Harlow, 1963] was the first to introduce marker particles to identify the domain of the fluid, which allowed particles to perform velocity advection in a Lagrangian manner by interpolating velocities between the particles and the grid. This frequent interpolation caused crippling numerical dissipation[3], which hindered the method's popularity. Thus to reduce the dissipation artifact, Zhu and Bridson [2005] introduced the fluid-implicit particle (FLIP) method to the computer graphics community. This method interpolates velocity differences instead of whole velocity components. As a result, the visible dissipation artifacts of PIC are eliminated at the cost of introducing a slight noise instability that can be reduced by blending between PIC and FLIP interpolation methods. In the most recent iteration, Jiang et al. [2015] improved on the PIC/FLIP method by correcting the transfer of velocity information between particle and grid in a more physically accurate manner that conserves angular momentum. Ubiquity of these methods motivates a scheme that couples well with particle-based velocity advection. However, since our work is not focused on velocity advection schemes we employ a simpler

[2]Advection refers to the transfer of an internal fluid quantity like temperature, position or even velocity itself through the velocity field. This is defined more precisely in Chapter 2.

[3]Numerical dissipation refers to artificial smoothing of the discrete velocity in the spatial domain as a side effect of interpolation.

semi-Lagrangian scheme [Stam, 1999] in our simulations, which we describe in the next chapter. We emphasize that because particle advection is usually decoupled from the main incompressibility solve, we are not limiting our solution to the specific velocity advection method. In fact, since our choice of advection uses the standard simulation grid, it is immediate to extend our solver to use a particle advection method.

Particle methods can also be used to propagate the body of the fluid through the simulation grid. However, moving the body of the fluid can be achieved with other methods, completely decoupled from velocity advection. For instance, in liquid simulations, what we typically see is the liquid surface. This suggests that it is sufficient to employ a scheme to track the surface geometry of the liquid for each time step achieving animation. Indeed, many methods for surface tracking have been developed for this very purpose. Notably the work by Brochu and Bridson [2009] introduced a robust grid-free surface tracker called *El Topo* that resolves topology changes while maintaining a high quality triangle mesh. Following, a method, introduced recently [Chentanez et al., 2015], improved on the performance of El Topo albeit at the cost of permitting some overlapping geometry. Other surface tracking methods rely on a regular grid and marching squares (cubes in 3D) templates to reconstruct the evolving Lagrangian surface to resolve topological changes [Wojtan et al., 2009]. We follow a similar grid-based method [Müller, 2009] that reconstructs the surface when potential self-intersections are detected. In fact, to simplify the algorithm, we reconstruct the surface at every time step using only the core marching squares templates, ignoring special case geometries on the interior of cells. We use a simple clipping technique to resolve features where the liquid meets the solid as described in Chapter 5.

Finally we aim to model the effects of surface tension. Much work has been done to model surface tension forces using explicit time integration. Earlier work focused on a level-set based surface representation [Enright et al., 2003; Losasso et al., 2004], with additional work focused on specific phenomena such as contact angles [Wang et al., 2005] and bubbles [Hong and Kim, 2005]. More recently, more attention was given to explicit surface meshes in universal solvers [Clausen et al., 2013] that can model hydrophobic surfaces, or solvers targeted at specific phenomena like droplet pinch-off [Thürey et al., 2010]. Unfortunately a basic explicit surface tension formulation imposes a restriction on the admissible time step $\Delta t \in O(\Delta x^{\frac{3}{2}})$ [Brackbill et al., 1992]. A great difficulty in removing this restriction is a lack of an explicit surface representation, which is typical in schemes using implicit surfaces [Zheng et al., 2009; Sussman and Ohta, 2009; Liovic et al., 2010]. A semi-implicit surface tension formulation was developed on explicit surface meshes independently by Schroeder et al. [2012] and Misztal et al. [2012] which happen to be equivalent as we will see in Chapter 4. Later, the work of Schroeder et al. was extended [Zheng et al., 2015] by coupling the pressure solve on the MAC grid with an explicit triangulated mesh near the

surface of the liquid using particles, although the work of Schroeder et al., which coupled an explicit surface mesh to the MAC grid, more closely resembles our work. However, by taking a different approach to pressure projection, we are able to handle cells with arbitrary complexity on the surface, without needing to invent complex interpolation operators between non-conforming Eulerian and Lagrangian meshes. Our formulation is also simpler than the work of Zheng et al. because it does not require auxiliary particles or complex triangulations near the surface mesh. Particles can be introduced, however, for the purposes of advection as mentioned earlier.

1.2 Contributions

We present a novel approach to simulating water-like liquids with a surface tension model able to simulate specific contact angles. Introducing the use of the mimetic finite difference (MFD) method to inviscid incompressible free surface flows (water-like liquids) with accurate boundary conditions, gives us the flexibility to build an Eulerian grid-based method seamlessly coupled to a Lagrangian surface mesh. This further allows us to add a semi-implicit surface tension scheme [Misztal et al., 2012], which we adapt to the MFD discretization and extend to handle specific contact angles. Our discretization can be seen as an extension to existing MAC based finite difference pressure solvers, by modifying the discrete Laplacian near the surface with a modified version prescribed by the MFD method augmented with semi-implicit surface tension. The individual contributions can be listed as follows:

- application of the mimetic finite difference method to water-like liquids,
- coupling a semi-implicit surface tension formulation to the MFD pressure solve,
- extending the surface tension formulation to the MFD discretization with handling of specific contact angles and
- meshing techniques required to make the above methods stable.

Our extension of the MAC based pressure projection to the surface of the liquid is especially simple to implement as we will see in Chapter 3.

Chapter 2

Fluid Equations

Water is a fluid, and in the most common setting, fluid motion is governed by the incompressible Navier-Stokes equations, typically referred to as momentum and incompressibility equations, respectively:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{f} + \nu\nabla \cdot \nabla\vec{u}$$
$$\nabla \cdot \vec{u} = 0$$

where $\vec{u}$ is the velocity field, p is the pressure driving the internal source term ∇p, ρ is fluid density[1], and $\vec{f}$ is the external source term representing external forces (e.g. gravity). $D\vec{u}/Dt$ is the total derivative of velocity given by

$$\frac{D\vec{u}}{Dt} = \frac{\partial\vec{u}}{\partial t} + \vec{u} \cdot \nabla\vec{u},$$

which describes how the velocity of a fluid particle changes as it moves with the flow. Finally, ν is the kinematic viscosity. It measures the viscosity of the fluid. We divert the interested reader to seek the formulation of these equations in a well-established source [Chorin et al., 1990]. We will use this model as a basis for our computational method.

Recall that we are interested in simulating water-like liquids, and generally speaking these have very low viscosity. We opt to ignore viscosity in our model altogether, because our simulations typically suffer from some numerical dissipation due to a heavy use of interpolation during the velocity advection stage of our simulator as we will see later. This

[1]Note that fluid density ρ is constant since the fluid is incompressible.

causes damping in fluid motion, which can be interpreted as the missing physical viscosity. Thus, we will focus our study on *inviscid*[2] fluids, so we can ignore the viscosity term in the momentum equation. These Navier-Stokes equations without viscosity are called the *incompressible Euler equations*.

2.1 Incompressible Free Surface Flow

As we have established, in order to simulate a water-like fluid, we need to solve the incompressible Euler equations given by:

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla p = \vec{f} \qquad \text{(momentum)} \tag{2.1a}$$

$$\nabla \cdot \vec{u} = 0 \qquad \text{(incompressibility)} \tag{2.1b}$$

To represent real liquids, these equations should be solved on a three-dimensional spacial domain $\Omega \subset \mathbb{R}^3$ representing the body of the fluid. However we will focus on a simplification of this problem to two dimensions, so we assume $\Omega \subset \mathbb{R}^2$. This means that the functions $\vec{u}$, $\vec{f}$, and p are defined on the domain $[0,\infty) \times \Omega$ with time parameter $t \in [0,\infty)$ and spacial parameters $(x,y) \in \Omega$. To be precise, $\vec{u}, \vec{f} : [0,\infty) \times \Omega \to \mathbb{R}^2$ and $p : [0,\infty) \times \Omega \to \mathbb{R}$. Naturally, the gradient and divergence operators (∇ and $\nabla\cdot$ resp.) correspond to derivatives in the spacial domain and can be written in the standard basis as

$$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right)^\top \qquad \text{and} \qquad \nabla \cdot \vec{u} = \frac{\partial \vec{u}}{\partial x} + \frac{\partial \vec{u}}{\partial y}.$$

Unsurprisingly, the momentum equation (2.1a) describes how the fluid accelerates due to forces acting on it. It is simply Newton's second law applied to fluids. The second term on the left-hand side describes the forces due to pressure imbalance inside the fluid, and the right hand side is responsible for all external forces like gravity and surface tension. The incompressibility equation (2.1b) ensures that the fluid cannot be compressed. This is an idealization, since even water is compressible at high pressures, but this effect is typically not observable in the absence of shock-waves. In fact, some particle based methods for liquid simulation like SPH, suffer from excessive compressibility, so researchers actively develop better methods [Solenthaler and Pajarola, 2009; Ihmsen et al., 2014a] to battle this effect to make liquids appear more natural. A fluid is incompressible if its volume remains

[2] An idealized fluid with no viscosity is called *inviscid*.

constant throughout the simulation. This is true when at every point in time, the amount of fluid entering the fluid domain is the same as the amount leaving. Mathematically, this means that the velocity field $\vec{u}$, normal to the liquid boundary, integrates to zero on the boundary:

$$\int_{\partial\Omega} \vec{u} \cdot \hat{n} = 0,$$

where $\hat{n}$ is the outward normal to the liquid domain Ω. By the divergence theorem [Chorin et al., 1990] this is equivalent to $\int_\Omega \nabla \cdot \vec{u} = 0$. Since this condition must hold in every part of the fluid, i.e. for any $V \subset \Omega$, it is necessary that $\nabla \cdot \vec{u} = 0$, which is precisely (2.1b). In other words, the velocity field must be divergence free.

To simplify the problem further, we will separate (2.1) into three parts:

$$\frac{D\vec{u}}{Dt} = 0 \qquad \text{(advection)} \tag{2.2a}$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{f}_1 \qquad \text{(body forces)} \tag{2.2b}$$

$$\frac{\partial \vec{u}}{\partial t} + \frac{1}{\rho}\nabla p = \vec{f}_2 \qquad \text{s.t.} \qquad \nabla \cdot \vec{u} = 0 \qquad \text{(projection)} \tag{2.2c}$$

where external body forces are separated into two parts: $\vec{f} = \vec{f}_1 + \vec{f}_2$, because it can be beneficial to apply implicit or semi-implicit methods to couple certain forces with the projection step. The method of breaking partial differential equations (PDEs) into parts (as we split (2.1) into (2.2)) is typically referred to as *operator splitting*. Splitting gives us, at best, a first order velocity approximation in time, which is sufficient for our purposes, and is the standard method used for fluid simulation in computer graphics. This enables us to use different integration schemes at each step, and ultimately exploit efficient ways to solve each equation. The simplicity and flexibility of this approach typically outweighs any advantages we get from tackling (2.1) with one monolithic method, at least for animation. Bridson [2015, §2.1] describes splitting in more detail and explains why it's prevalent in fluid simulations.

Discretizing the three equations (2.2) in time, effectively forms our time stepping algorithm used to animate the motion of the liquid. Let's fix Δt as the time step parameter at the start of the simulation. At each step we start with some velocity field $\vec{u}(t) : \Omega \to \mathbb{R}^2$, fixed at some time t, and we solve (2.2) to find the incompressible velocity field for the next time step, $\vec{u}(t + \Delta t)$, which we use to move the liquid. First, we solve the advection equation (2.2a), with $\vec{u}(t)$ as the current velocity field, giving us an intermediate velocity field $\vec{u}^*(t + \Delta t)$. Then, we apply body forces (like gravity) by solving equation 2.2b using a simple explicit forward Euler integration scheme, and $\vec{u}^*(t + \Delta t)$ as the current velocity field.

This gives us the second intermediate velocity field $\vec{u}^{**}(t+\Delta t)$. Finally, using $\vec{u}^{**}(t+\Delta t)$, we discretize the projection equation (2.2c) in time, to form an elliptic PDE, which we subsequently solve by projecting $\vec{u}^{**}(t+\Delta t)$ onto a nearby incompressible velocity field, giving us $\vec{u}(t+\Delta t)$. At the end of each step we increment the time variable by Δt. In the following subsections we will describe the time discretization schemes used for each equation in (2.2) in more detail.

2.1.1 Velocity Advection

The velocity is advected by solving equation 2.2a on a divergence-free velocity field. We use a standard unconditionally stable semi-Lagrangian advection scheme first introduced to graphics by Stam [1999]. There have been a number of methods developed to solve for advection, including particle methods, which bypass having to solve equation 2.2a altogether. Since advection is not the main focus of our research, we will only briefly describe semi-Lagrangian advection for completeness.

We use a standard Marker-And-Cell (MAC) type grid of staggered velocities, which is depicted in Figure 2.1. This velocity grid coincides with our discretization for pressure projection described in a later section. More precisely, vertical and horizontal velocity samples are stored separately at midpoints of grid edges, with one sample per edge. We are purposely vague on the exact location of these samples as it is currently not important, but will be described more precisely later.

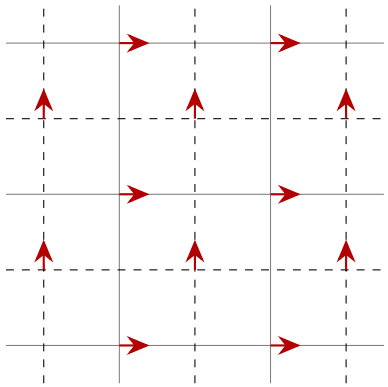


Figure 2.1: Staggered velocity grid with horizontal velocity components are sampled on the solid grid and the vertical velocity components are sampled on the dashed grid.

Recall that equation 2.2a implies that the velocity of a fluid particle does not change as it moves along the velocity field. This means that if we consider a particle at a grid node, its velocity is the same as it was in the previous time step. Mathematically, suppose $\vec{u}_{i,j}(t)$ is a velocity component sampled at position $\vec{x}_{i,j}$ on row i and column j of a regular grid at time t. Then, at time $t+\Delta t$, a fluid particle in position $\vec{x}_{i,j}$ must have travelled from some position that can be approximated by $\vec{x}_{prev} = \vec{x}_{i,j} - \Delta t\vec{u}_{i,j}(t)$ (higher order time schemes are also possible). Thus the new velocity $\vec{u}^{*}_{i,j}(t+\Delta t)$ on $\vec{x}_{i,j}$, is the velocity sampled at $\vec{x}_{prev}$ from the previous time step. Since $\vec{x}_{prev}$ may not lie on a grid node, we need to interpolate

the velocities from the neighbouring grid locations and set

$$\vec{u}^*_{i,j}(t + \Delta t) = \texttt{interpolate}(\vec{u}_{i,j}(t), \vec{x}_{prev}).$$

Note that we can use a more accurate time integration scheme to compute $\vec{x}_{prev}$, and any reasonable interpolation scheme can work for `interpolate`. Furthermore, note that grid velocity samples are given from the previous time step, and may not be present near $\vec{x}_{prev}$. Thus prior to the velocity advection step we need to extrapolate velocities to outside of the liquid domain [Bridson, 2015, Ch. 4]. By extrapolating the velocity field, we ensure that velocity samples are available everywhere for the interpolation step.

Unfortunately semi-Lagrangian advection suffers from significant numerical dissipation, which can be interpreted as an approximation of viscosity. However, simulations like the crown splash of a drop of milk look overly damped. Luckily there are a body of sophisticated advection techniques available in computer graphics to ameliorate this issue as discussed in Section 1.1. One close relative of the semi-Lagrangian method that has not been mentioned before is semi-Lagrangian-MacCormack advection. This method does one regular semi-Lagrangian advection step just described, and corrects the error using a backward semi-Lagrangian step with negated velocities. The details of this method are thoroughly examined and compared with the standard semi-Lagrangian scheme in [Selle et al., 2008]. As mentioned before, particle advection methods like the Particle-in-Cell (PIC) [Harlow, 1963] or the Fluid-Implicit-Particle (FLIP) [Brackbill and Ruppel, 1986] and their derivatives [Zhu and Bridson, 2005; Ferstl et al., 2016] can also be used with our method, however more work is required to couple particles and our explicit Lagrangian surface.

2.1.2 Body Forces

Solving equation 2.2b is typically done with a simple explicit forward Euler integration scheme:

$$\vec{u}^{**}(t + \Delta t) = \vec{u}^*(t + \Delta t) + \Delta t \vec{f}_1(t),$$

which gives us the second intermediate velocity field, $\vec{u}^{**}(t + \Delta t)$. In general, $\vec{f}_1$ can include gravity, surface tension and other external forces from other dynamic objects. However, treating surface tension and pressure projection separately forces a stiff time-step restriction as we will see in Chapter 4. Thus we will include surface tension in the pressure projection step, so $\vec{f}_1$ consists solely of gravitational acceleration $\vec{g}$.

2.1.3 Pressure Projection

Finally we need to ensure that the velocity field used to move our fluid is divergence free. Thus it remains to project the intermediate velocity field onto a divergence free velocity field by solving equation 2.2c. For the sake of exposition, we first assume $\vec{f}_2 = 0$, and reintroduce surface tension in a later chapter. Since density is constant, we can assume $\rho = 1$ without loss of generality. We now discretize the time derivative using the backward Euler scheme to get equations

$$\vec{u}(t + \Delta t) = \vec{u}^{**}(t + \Delta t) - \Delta t \nabla p(t + \Delta t) \tag{2.3a}$$

$$\nabla \cdot \vec{u}(t + \Delta t) = 0 \tag{2.3b}$$

We can further simplify these equations by eliminating the velocity unknown:

$$\Delta t \nabla \cdot \nabla p(t + \Delta t) = \nabla \cdot \vec{u}^{**}(t + \Delta t). \tag{2.4}$$

This equation is a standard elliptic PDE, referred to as Poisson's problem where $\nabla \cdot \vec{u}^{**}(t+\Delta t)$ is the source term. From this point forward, to simplify our equations, we will assume that the pressure and velocity fields are evaluated at time $t + \Delta t$ unless otherwise stated. Thus pressure p and velocities $\vec{u}^{**}$ and $\vec{u}^{*}$ can be seen to be defined solely in the spatial dimensions.

Boundary Conditions

We have not yet mentioned what happens to our fluid at the boundary where the fluid meets air or a solid wall. This is a critical part of simulating liquids for animation since the liquid-air interface is the one we see, and the point where the three media (liquid, solid and air) meet shows whether the liquid beads up or flattens out due to surface tension. More precisely, at the liquid-air interface the pressure is a fixed constant, because liquids in a common setting are exposed to an effectively constant atmospheric pressure. At the liquid-solid interface, the pressure under a resting body of liquid, for instance, depends on the height of liquid above it, so we cannot constrain it to a constant. However, we can assume that the liquid either sticks to the solid surface (a standard assumption in continuous fluid mechanics) or it moves freely along the surface. In either case, we assume that solids are impermeable. This property can be enforced with a no-penetration boundary condition that matches the velocity normal to the surface of the solid, with the velocity of the fluid. The constraints on the velocity tangential to the solid surface are referred to as no-slip (setting tangential component to zero) and free-slip (unconstrained tangential

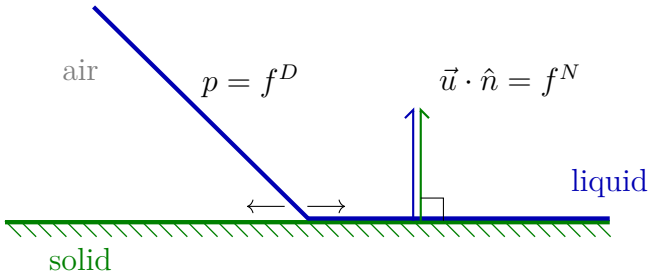


Figure 2.2: No-penetration boundary condition forces the velocity of the liquid surface match that of the touching solid surface in the normal direction. Free-slip allows the liquid to slide along the solid surface freely.

component) boundary conditions. We will focus on the free-slip condition as it is the natural condition for the inviscid case.

We handle behaviour at these interfaces by imposing boundary conditions on the PDE (2.4):

$$p = f^D \text{ on } \Gamma^D \qquad \text{(Dirichlet)} \tag{2.5a}$$

$$\hat{n} \cdot \vec{u} = f^N \text{ on } \Gamma^N \qquad \text{(Neumann)} \tag{2.5b}$$

where Γ^D and Γ^N are parts of the boundary on which Dirichlet and Neumann conditions are enforced respectively, and $\hat{n}$ is the outward facing normal to the domain boundary $\partial\Omega = \Gamma^D \cup \Gamma^N$. This is illustrated in Figure 2.2. Using equation 2.3a, the Neumann boundary condition can be rewritten in terms of pressure as

$$\hat{n} \cdot \Delta t \nabla p = \hat{n} \cdot \vec{u}^{**} - f^N \text{ on } \Gamma^N. \tag{2.6}$$

To enforce the constraints on the liquid interface, we set $f^D = 0$ to fix pressure at the liquid-air boundary and set $f^N = 0$ to enforce the free-slip condition along static solid walls. More precisely, we can set f^D to the atmospheric pressure, but this will only affect the solution for pressure in (2.4) up to a constant. However, in animation, we are only interested in velocities, and since the gradient of a constant is zero, equation 2.3a tells us that velocities remain unaffected. Furthermore, in order to simulate moving solid boundaries, it is also desirable to handle non-zero Neumann boundary conditions. In this case we will set $f^N = \hat{n} \cdot \vec{u}_{solid}$, where $\vec{u}_{solid}$ is the velocity of the solid object.

Note that for problems with pure Neumann boundary conditions (i.e. $\Gamma^N = \partial\Omega$) we

have

$$\begin{aligned}
\int_{\partial\Omega} \hat{n} \cdot \vec{u}^{**} &= \int_{\Omega} \nabla \cdot \vec{u}^{**} && \text{by the divergence theorem,} \\
&= \Delta t \int_{\Omega} \nabla \cdot \nabla p && \text{by (2.4),} \\
&= \Delta t \int_{\partial\Omega} \hat{n} \cdot \nabla p && \text{by the divergence theorem,} \\
&= \int_{\partial\Omega} \hat{n} \cdot \vec{u}^{**} - f^N && \text{by (2.6).}
\end{aligned}$$

This compatibility condition requires that f^N must be zero-mean:

$$\int_{\partial\Omega} f^N = 0.$$

Since we do not simulate air, we will not encounter pure Neumann boundary conditions in our liquid simulations. This condition is significant, however, when conducting convergence tests to ensure that boundary conditions are set correctly (see Sections 6.1 and 6.2 for numerical convergence tests).

2.2 Surface Tracking

So far we have outlined the components of a standard fluid simulator, while deliberately omitting the details of spatial discretization for pressure projection, since it will be covered in the next chapter.

In order to build a working simulator, we need a means to represent the motion of a body of liquid along the velocity field we worked so hard to compute from the incompressible Euler equations. The simplest and most intuitive method to describe a liquid is with particles moving through space. Then we can simply move each particle i at position $\vec{x}_i$ through the computed velocity field $\vec{u}$ to the new position

$$\vec{x}_i^{\,new} = \vec{x}_i + \Delta t \vec{u}(\vec{x}_i). \tag{2.7}$$

As we have already mentioned, there are other ways to represent the liquid in an Eulerian based fluid simulator, but using an explicit Lagrangian surface mesh is advantageous when dealing with surface tension effects. Thus instead of particles on the interior of the liquid, we will represent only the surface of the liquid with a closed curve (or a set of closed curves) connecting the particles on the surface. More specifically, we will connect surface particles with line segments, and move them as in equation 2.7. We leave the details of this approach to Chapter 5.

2.3 Overview

To summarize the elements of fluid simulation we propose the following general algorithm employed in our work.

Algorithm 1 UNIFIED-MFD

1: Initialize surface mesh with vertices $\vec{x}_i$. (Chapter 5)
2: Initialize velocity field $\vec{u}$.
3: **loop**
4: Build simulation mesh from surface vertices $\vec{x}_i$. (Chapter 5)
5: Advect velocity field $\vec{u}$.
6: Apply body forces (gravity): $\vec{u} \leftarrow \vec{u} + \Delta t \vec{g}$.
7: Make $\vec{u}$ incompressible. (Chapters 3 and 4)
8: Move surface mesh: $\vec{x}_i^{new} = \vec{x}_i + \Delta t \vec{u}(\vec{x}_i)$. (Section 5.1)
9: **print** Surface mesh defined by vertices $\vec{x}_i$.
10: **end loop**

Chapter 3

The Mimetic Finite Difference Method

The Mimetic Finite Difference (MFD) method is a framework used to solve a variety of partial differential equations (PDEs), not unlike the finite-volume (FV) and finite-element (FE) methods. Unlike standard FV and FE methods, however, the MFD method mimics some fundamental mathematical and physical properties including duality of differential operators, exact identities of vector and tensor calculus and conservation laws on a vast family of unstructured meshes. A comprehensive review of the MFD method can be found in [Lipnikov et al., 2014] and a more fundamental introduction in [Veiga et al., 2014].

To the best of our knowledge, the MFD machinery has not been applied to simulation in computer graphics, although it has been applied to geometry processing [Alexa and Wardetzky, 2011]. This chapter gently introduces the MFD method as it is applied in our specific context to solve the Poisson problem in (2.4).

Let $\Omega \subset \mathbb{R}^2$ be our two-dimensional computational domain on which the Poisson problem is discretized. We take Ω to be polygonal with Lipshitz continuous boundary. Further, consider an unstructured non-overlapping decomposition of Ω into polygonal cells $c \subset \Omega$. These cells need not be convex, however they must satisfy a regularity condition in order to guarantee convergence of the MFD method under mesh refinement. We refer to this decomposition as a polygonal mesh. A mesh is said to be regular if there exist fixed quantities $N_* \in \mathbb{Z}$ and $\alpha_* \in \mathbb{R}$, such that the following condition holds:

Condition 3.0.1 (Regularity)**.** Each polygon can be split into at most N_* triangles t, such that

$$\frac{r_t}{d_t} > \alpha_*, \tag{3.1}$$

where r_t is the radius of the inscribed circle, and d_t is the length of the largest edge of t.

This assumption permits a large family of shapes, including non-convex ones. Not unlike FE and FV methods, we are constrained by mesh quality. However, no explicit triangulation of the cells is required; we only require that such a triangulation *is possible*. This regularity condition is similar for 3D MFD discretizations where a *polyhedral* cell is partitioned into *tetrahedra* with a bound on the ratio between the radius of the inscribed *sphere* and the diameter for each tetrahedron.

It is worth noting that the ratio in (3.1) is also used to measure the quality of tetrahedral meshes used for FE simulations [Alliez et al., 2005; Shewchuk and Si, 2014], and is equivalent to other measures like the minimum and maximum dihedral angles [Bronson et al., 2014]. Creating good quality tetrahedral meshes has been an active topic of research [Alliez et al., 2005; Labelle and Shewchuk, 2007; Bronson et al., 2014; Shewchuk and Si, 2014; Si, 2015] to this day. Naturally, using the FE method on an existing mesh with convex cells would require an explicit subdivision of cells, while the MFD method works on polygonal meshes out of the box. Furthermore, this subdivision would have to maximize the ratio in the regularity condition, which is typically a non-trivial task. Alternatively, it is possible to extend the FE method to a larger set of admissible cells as to avoid explicit triangulation, however this becomes more complicated and does not scale well with the number of unique cell shapes. Thus one advantage of the MFD approach to FE methods is not having to maintain an explicit high quality triangle/tetrahedron mesh.

The Mimetic Finite Difference method is a close cousin of the Finite Volume method. Notably, both methods can handle irregular elements, and both methods aim to be conservative. For instance in cell centred discretizations, the flux between cells is balanced by the discrete divergence operator. The main difference between the two methods is the approximation of constitutive laws. Fourier's law [Halley, 2006, p. 199], for instance, requires the heat transfer rate to be proportional to the negative temperature gradient and the area orthogonal to the flow. The line connecting cell centres on an unstructured mesh may not always be orthogonal to the interface between the cells. Since the FV method draws the discrete gradient from the centres of adjacent cells, it may not satisfy orthogonality. Hence basic FV methods are often used with Voronoi meshes that satisfy this orthogonality constraint by definition. These issues have been ameliorated by extensions on the FV formulation, e.g. Discrete Duality Finite Volume (DDFV) [Domelevo and Omnes, 2005] and Hybrid or Mixed Finite Volume (HFV, MFV) [Eymard et al., 2007; Thomas and Trujillo, 1999] methods. However, these extensions require auxiliary meshes or additional degrees of freedom. The MFD method relies on the construction of a discrete inner product that provides discrete duality between the gradient and divergence operators (similar to DDFV).

3.1 Discretization

In this section, we will describe one particular discretization of the pressure and velocity functions on the discrete mesh. First, we need to describe the geometry of the mesh in more detail.

3.1.1 Domain Discretization

In 2D, we can describe the mesh as a planar directed graph embedded into $\mathbb{R}^2$. As such, let N_V denote the total number of mesh nodes, which are located at positions $\vec{x}_n \in \mathbb{R}^2$. Edges are identified by a pair of node indices $e = (n, m)$, and cells are denoted by c as mentioned before. Figure 3.1a highlights the different components of the mesh. We will denote the boundary of each cell by ∂c. Furthermore, each cell boundary is a disjoint union of edge sets denoted by

$$\phi(e) = \{\, t\vec{x}_n + (1-t)\vec{x}_m \;:\; t \in [0, 1) \,\}.$$

By a slight abuse of notation, we will use e and $\phi(e)$ somewhat interchangeably and simply write $e \in \partial c$ to mean $\phi(e) \subset \partial c$, because for our purposes it is only important to note when an edge e is on the boundary of c or Ω.

We will also need to draw a discrete analog to the continuous problem. In this setting we will enumerate our nodes, cells and edges. Thus let N_E and N_C denote the numbers of edges and cells respectively, and reuse the symbol c to index cells from 1 to N_C, and the symbol e to index edges from 1 to N_E.

The meaning of symbols c and e will be clear from the context. For instance, appearing as a domain of integration in $\int_e$, e is synonymous to $\phi(e)$, while appearing as an index into a vector in $\mathbf{u}_e$, e is synonymous to an index between 1 and N_E. This notation allows us to draw a clear connection between the continuous problem and its discretization seamlessly.

Now we can describe the notation used to describe the variables involved in solving Poisson's problem. Pressure is a scalar function $p : \Omega \to \mathbb{R}$, and velocity is a vector function $\vec{u} : \Omega \to \mathbb{R}^2$. Discrete pressure values are stored on nodes. Velocities are stored on edge midpoints as tangential components of $\vec{u}$ in a *fixed* direction of the edge. In 2D, edges correspond to interfaces between cells. The pressure degrees of freedom are denoted by $p_n = p(\vec{x}_n)$ for each node n. The vector describing these pressure values at all nodes is given by $\mathbf{p}_n = p_n$, hence $\mathbf{p} \in \mathbb{R}^{N_V}$. Sampled velocities are given by $u_e = \frac{1}{|e|}\int_e \vec{u} \cdot \hat{\tau}_e \, dL$ for each edge $e = (n, m)$, where $\hat{\tau}_e = (\vec{x}_m - \vec{x}_n)/|e|$ is the tangential unit vector along the edge and $|e| = \|\vec{x}_m - \vec{x}_n\|$. The vector describing these velocity values at all edges on the discrete

mesh is denoted by $\mathbf{u}_e = u_e$, hence $\mathbf{u} \in \mathbb{R}^{N_E}$. An example of such a discretization is shown in Figure 3.1b.

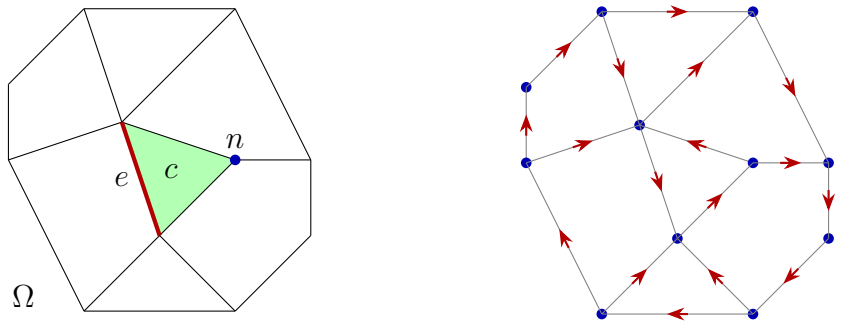


Figure 3.1: (a) Decomposition of the domain Ω into cells c, with edges e, and nodes n. (b) Node-edge based discretization of the domain using the mesh in (a). Velocity components u_e are stored at the midpoints of the edges (arrows). Pressures p_n are stored at the nodes (dots).

3.1.2 Discrete Operators

We define a discrete gradient operator $\mathbf{G} : \mathbb{R}^{N_V} \to \mathbb{R}^{N_E}$ on the mesh by

$$(\mathbf{Gp})_e = \frac{p_m - p_n}{\|\vec{x}_m - \vec{x}_n\|}. \tag{3.2}$$

Note that $\mathbf{G}$ is a sparse matrix of positive and negative values. In particular, each row corresponds to an edge (n, m) with a negative value for the starting node n, and a positive value for the ending node m.

Example 3.1.1. The discrete gradient matrix on a simple quadrilateral cell is given by the following matrix with rows representing edges in the order: *left*, *bottom*, *right* and *top*, with lengths l, b, r and t respectively directed as depicted in Figure 3.2.

$$\mathbf{G}^{\square} = \begin{pmatrix} -l^{-1} & l^{-1} & 0 & 0 \\ -b^{-1} & 0 & 0 & b^{-1} \\ 0 & 0 & r^{-1} & -r^{-1} \\ 0 & -t^{-1} & t^{-1} & 0 \end{pmatrix}$$

Assuming the order of the vertices follows in the clockwise direction starting from the *bottom-left* node.

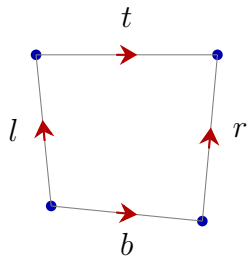


Figure 3.2: Example of quadrilateral cell.

This discrete gradient operator is identical to the one found in standard finite difference and finite volume schemes. The MFD method deviates, however, from other methods at the introduction of a corresponding discrete divergence operator, which we define as the dual of the given gradient operator:

$$\mathbf{D} = -\mathbf{M}_V^{-1}\mathbf{G}^\top\mathbf{M}_E, \tag{3.3}$$

where $\mathbf{M}_V$ and $\mathbf{M}_E$ are some symmetric positive definite (SPD) matrices that we will define later. Let us define two inner products on $\mathbb{R}^{N_V}$ and $\mathbb{R}^{N_E}$ as follows

$$\langle \mathbf{q}, \mathbf{r} \rangle_V = \mathbf{q}^\top \mathbf{M}_V \mathbf{r} \qquad \forall \mathbf{q}, \mathbf{r} \in \mathbb{R}^{N_V} \tag{3.4}$$

$$\langle \mathbf{v}, \mathbf{w} \rangle_E = \mathbf{v}^\top \mathbf{M}_E \mathbf{w} \qquad \forall \mathbf{v}, \mathbf{w} \in \mathbb{R}^{N_E}. \tag{3.5}$$

Then the duality between $\mathbf{G}$ and $\mathbf{D}$ holds by construction, because we have

$$\langle \mathbf{G}\mathbf{q}, \mathbf{v} \rangle_E = -\langle \mathbf{q}, \mathbf{D}\mathbf{v} \rangle_V \qquad \forall \mathbf{v} \in \mathbb{R}^{N_E}, \quad \forall \mathbf{q} \in \mathbb{R}^{N_V}, \tag{3.6}$$

which is analogous to the duality of the continuous gradient and divergence operators (see Appendix A). It follows from the definition of an inner product that $\mathbf{M}_E$ and $\mathbf{M}_V$ are SPD matrices. This formulation allows us to discretize the PDE in (2.4) as a symmetric positive semi-definite system $\Delta t \mathbf{D}\mathbf{G}\mathbf{p} = \mathbf{D}\mathbf{u}^{**}$, which reduces to

$$\Delta t \mathbf{G}^\top \mathbf{M}_E \mathbf{G}\mathbf{p} = \mathbf{G}^\top \mathbf{M}_E \mathbf{u}^{**}, \tag{3.7}$$

by multiplying both sides by the diagonal matrix $\mathbf{M}_V$ (see Appendix A for the derivation of $\mathbf{M}_V$). Solving this system will give us a pressure field $\mathbf{p}$, which we can use to compute the divergence-free velocity field as

$$\mathbf{u} = \mathbf{u}^{**} - \Delta t \mathbf{G}\mathbf{p}, \tag{3.8}$$

which in effect removes the curl free part of the Helmholtz-Hodge decomposition [Chorin et al., 1990, pp. 36-37] from $\mathbf{u}^{**}$ and leaves us with divergence free velocities. Using equation 3.8 and substituting it into (3.7), we can alternatively solve the system

$$\begin{aligned} \mathbf{M}_E\mathbf{u} + \Delta t\mathbf{M}_E\mathbf{G}\mathbf{p} &= \mathbf{M}_E\mathbf{u}^{**} \\ \Delta t\mathbf{G}^\top\mathbf{M}_E\mathbf{u} &= 0, \end{aligned}$$

for $\mathbf{u}$ and $\mathbf{p}$ which corresponds to a discretization of the Poisson's problem in mixed form as in equations 2.3. We define a pseudo-divergence operator $\tilde{\mathbf{D}} = \mathbf{G}^\top\mathbf{M}_E$, and let $\lambda = \Delta t\mathbf{p}$. Then we can write this system in block matrix form as

$$\begin{pmatrix} \mathbf{M}_E & \tilde{\mathbf{D}}^\top \\ \tilde{\mathbf{D}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{M}_E\mathbf{u}^{**} \\ \mathbf{0} \end{pmatrix}, \tag{3.9}$$

which is non-singular in the presence of a Dirichlet boundary condition (see Section 3.2). Solving this system gives the divergence free velocity field $\mathbf{u}$ directly. The symmetric matrix above conveniently corresponds to the Karush-Kuhn-Tucker (KKT) matrix [Nocedal and Wright, 2006, §16.1] used to solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\mathbf{u}^\top\mathbf{M}_E\mathbf{u} - \mathbf{u}^\top\mathbf{M}_E\mathbf{u}^{**} \\ \text{over} \quad & \mathbf{u} \in \mathbb{R}^{N_E} \\ \text{subject to} \quad & \tilde{\mathbf{D}}\mathbf{u} = \mathbf{0} \end{aligned} \tag{3.10}$$

as prescribed by the KKT conditions where λ acts as a Lagrange multiplier enforcing the equality constraints. In general, the system (3.9) is known to be indefinite [Nocedal and Wright, 2006, Ch. 16], and is more difficult and costly to solve than (3.7). However, this formulation will also allow us to include a semi-implicit scheme for surface tension more easily (see Section 4.2). Furthermore, by adding a constant term, we can rewrite the objective function in (3.10) as

$$\frac{1}{2}(\mathbf{u} - \mathbf{u}^{**})^\top\mathbf{M}_E(\mathbf{u} - \mathbf{u}^{**}), \tag{3.11}$$

which can also be written more concisely as

$$\frac{1}{2}\|\mathbf{u} - \mathbf{u}^{**}\|^2_{\mathbf{M}_E}, \tag{3.12}$$

where $\|\cdot\|_{\mathbf{M}_E}$ denotes the inner product norm from (3.5). This form directly shows that we are looking for a divergence-free projected velocity field that is closest to $\mathbf{u}^{**}$. Note that the matrix $\mathbf{M}_V$ does not appear in either formulation, so we can safely ignore it and dropping the subscript E, rewrite $\mathbf{M} = \mathbf{M}_E$ and naturally $\langle\cdot,\cdot\rangle = \langle\cdot,\cdot\rangle_E$.

Building the mass matrix M

We assemble the matrix $\mathbf{M}$ on a per-cell basis mimicking the additivity of the continuous inner product:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_c \langle \mathbf{u}_c, \mathbf{v}_c \rangle_c = \sum_c \mathbf{u}_c^\top \mathbf{M}_c \mathbf{v}_c \quad \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^{N_E} \tag{3.13}$$

where $\mathbf{u}_c$ and $\mathbf{v}_c$ are the vectors $\mathbf{u}$ and $\mathbf{v}$ restricted to the edges of cell c, and $\mathbf{M}_c$ is the corresponding cell-based inner product matrix defining the local mimetic inner product $\langle \cdot, \cdot \rangle_c$. The MFD framework provides second order convergence of the pressure (with respect to grid resolution) under the following condition:

Condition 3.1.2 (Consistency)**.** Let $\mathbf{u}_c$ and $\mathbf{v}_c$ be the discrete representations of continuous vector fields $\vec{u}$ and $\vec{v}$ restricted to the cell $c \subset \Omega$ respectively. Furthermore, assume that $\vec{v}$ and $\nabla \times \vec{u}$ are constant in c and $\vec{u} \cdot \hat{\tau}_e$ is constant on each edge $e \in \partial c$. Then, we have an exact equality:

$$\langle \mathbf{u}_c, \mathbf{v}_c \rangle_c = \int_c \vec{u} \cdot \vec{v} \, dV. \tag{3.14}$$

In other words, the discrete inner product should exactly compute the integral in (3.14) for a particular set of piecewise constant vector fields.

We would like to build a matrix $\mathbf{M}_c$ that satisfies this condition. Let $\{ \hat{e}_1, \hat{e}_2, \hat{e}_3 \}$ be an orthonormal basis of $\mathbb{R}^3$. Then assume that we have vector fields $\vec{u}$ and $\vec{v}$ as described above. Where appropriate, we let vectors in $\mathbb{R}^2$ be embedded in $\mathbb{R}^3$ with the third coefficient being zero. To avoid ambiguities, we emphasize that integrals in this section are taken over non-empty subsets of $\Omega \times \{0\}$, thus essentially ignoring the third coordinate. Since $\vec{v}$ is constant in $c \times \mathbb{R} \subset \mathbb{R}^3$, we can express it as

$$\vec{v} = v_x \hat{e}_1 + v_y \hat{e}_2$$

in $c \times \mathbb{R}$ where $v_x, v_y \in \mathbb{R}$ are constants. Then let

$$\begin{aligned} \vec{q}_1(x, y, z) &= (y - y_c)\hat{e}_3 \quad \text{and} \\ \vec{q}_2(x, y, z) &= (x_c - x)\hat{e}_3 \end{aligned}$$

be two vector functions on $\mathbb{R}^3$, where (x_c, y_c) is the centroid of cell c. Note that $\hat{e}_i = \nabla \times \vec{q}_i$ for $i \in \{ 1, 2 \}$, so we can write

$$\vec{v} = v_x \nabla \times \vec{q}_1 + v_y \nabla \times \vec{q}_2 = \nabla \times (v_x \vec{q}_1 + v_y \vec{q}_2).$$

Now let $\vec{q} = v_x\vec{q}_1 + v_y\vec{q}_2 = q_z\hat{e}_3$, where $q_z(x,y) = v_x(y-y_c) + v_y(x_c - x)$. Then observe that

$$\begin{aligned}\int_c \vec{u}\cdot\vec{v}\,dV &= \int_c \vec{u}\cdot\nabla\times\vec{q}\,dV \\ &= \int_c \vec{q}\cdot\nabla\times\vec{u}\,dV + \int_c \nabla\cdot(\vec{q}\times\vec{u})\,dV \qquad \text{by a cross product identity}\end{aligned}$$

where the first term is identically zero since $\nabla\times\vec{u}$ is constant and $\int_c \vec{q}\,dV = 0$. The second term reduces further

$$\begin{aligned}\int_c \nabla\cdot(\vec{q}\times\vec{u})\,dV &= \int_{\partial c}(\vec{q}\times\vec{u})\cdot\hat{n}\,dL \qquad \text{by the divergence theorem} \\ &= -\sum_{e\in\partial c}\int_e \vec{u}\cdot(\vec{q}\times\hat{n}_{c,e})\,dL \\ &= -\sum_{e\in\partial c}\int_e \vec{u}\cdot\alpha_e q_z\hat{\tau}_e\,dL.\end{aligned}$$

The last step follows from the relationship between the edge normal and our fixed edge directions to be $\alpha_e\hat{\tau}_e = \hat{e}_3\times\hat{n}_{c,e}$ where $\alpha_e\in\{-1,1\}$ is determined by the orientation of the outward facing normal $\hat{n}_{c,e}$ and the fixed edge direction $\hat{\tau}_e$. Finally, since $\vec{u}\cdot\hat{\tau}_e$ is constant on each edge, we can write

$$\int_c \vec{u}\cdot\vec{v}\,dV = -\sum_{e\in\partial c}\alpha_e\vec{u}\cdot\hat{\tau}_e\int_e q_z\,dL = -\sum_{e\in\partial c}\alpha_e u_e\int_e q_z\,dL.$$

Note that $\mathbf{u}_c$ collects all elements u_e in $\mathbf{u}$ where $e\in\partial c$. Let $\mathbf{r}_c$ be a vector indexed by edges $e\in\partial c$ in the same order as in $\mathbf{u}_c$ and defined as

$$(\mathbf{r}_c)_e = -\alpha_e\int_e q_z\,dL.$$

Then we can write

$$\int_c \vec{u}\cdot\vec{v}\,dV = \mathbf{u}_c^\top\mathbf{r}_c.$$

Thus it suffices to find $\mathbf{M}_c$ such that

$$\mathbf{u}_c^\top\mathbf{M}_c\mathbf{v}_c = \mathbf{u}_c^\top\mathbf{r}_c \tag{3.15}$$

in order to satisfy the consistency condition.

Remark 3.1.3. For any $\mathbf{u}\in\mathbb{R}^{N_E}$, we can find a vector field $\vec{u}$ that satisfies the assumptions in condition 3.1.2. In particular, there exists a vector field $\vec{u}$ such that for each cell c,

$$\nabla\times\vec{u} = \frac{1}{|c|}\sum_{e\in\partial c}\alpha_e|e|u_e \qquad \text{in } c, \text{ and} \tag{3.16}$$

$$\hat{\tau}_e\cdot\vec{u} = \alpha_e u_e \qquad \text{on each edge } e\in\partial c. \tag{3.17}$$

Here, $|c|$ is the area of the cell and $|e| = \|\vec{x}_m - \vec{x}_n\|$ for edge $e = (n, m)$ as before. The field $\vec{u}$ is a non-unique solution to the differential equation (3.16) with Neumann boundary conditions (3.17).

The remark above implies that $\mathbf{u}$ can be arbitrary since we can always find a corresponding $\vec{u}$. Hence the consistency condition 3.1.2 must be satisfied for an arbitrary $\mathbf{u}_c$. Thus we can drop $\mathbf{u}_c$ from (3.15), and write the consistency condition as $\mathbf{M}_c\mathbf{v}_c = \mathbf{r}_c$ for all possible $\mathbf{v}_c$. Now since it suffices to satisfy this condition on the basis of $\mathbb{R}^2$, we can write it as

$$\mathbf{M}_c\mathbf{N}_c = \mathbf{R}_c \tag{3.18}$$

where $\mathbf{N}_c$ and $\mathbf{R}_c$ are defined on each row indexed by e as

$$(\mathbf{N}_c)_e = (\hat{\tau}_e \cdot \hat{e}_1,\ \hat{\tau}_e \cdot \hat{e}_2) = \hat{\tau}_e^\top \tag{3.19a}$$
$$(\mathbf{R}_c)_e = \alpha_e|e|\,(y_c - y_e,\ x_e - x_c) \tag{3.19b}$$

where (x_e, y_e) is the midpoint of edge e. It can be shown that $\mathbf{R}_c^\top\mathbf{N}_c$ is symmetric positive definite [Veiga et al., 2014, Ch. 4]. Then the matrix

$$\mathbf{M}_c^0 = \mathbf{R}_c(\mathbf{R}_c^\top\mathbf{N}_c)^{-1}\mathbf{R}_c^\top \tag{3.20}$$

solves equation 3.18 since the inverse of $\mathbf{R}_c^\top\mathbf{N}_c$ exists. Unfortunately, $\mathbf{M}_c^0$ can be ill-conditioned and using it alone, can produce unstable results. Luckily, this is not the only solution to (3.18). The general solution can be written as

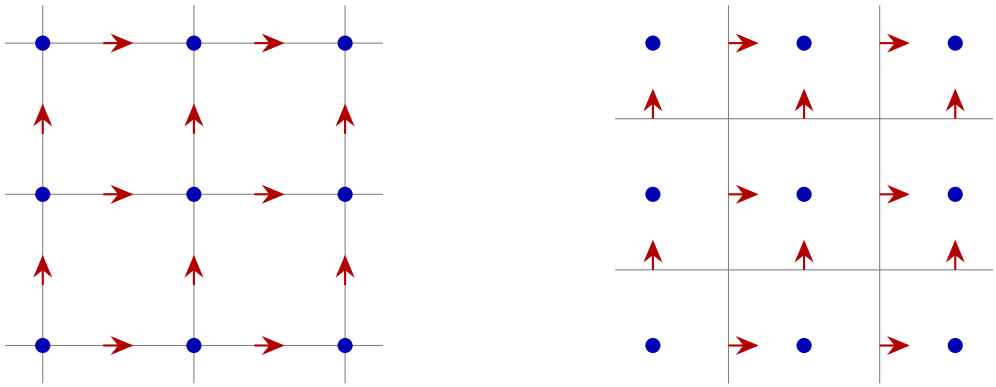
$$\mathbf{M}_c = \mathbf{M}_c^0 + \mathbf{M}_c^1 \tag{3.21}$$

where naturally $\mathbf{M}_c^1\mathbf{N}_c = \mathbf{0}$. We can impose a stability condition on our solution to alleviate this issue:

Condition 3.1.4 (Stability)**.** There exist two positive constants C_* and C^* independent of the mesh size, such that

$$C_*|c|\mathbf{v}_c^\top\mathbf{v}_c \le \mathbf{v}_c^\top\mathbf{M}_c\mathbf{v}_c \le C^*|c|\mathbf{v}_c^\top\mathbf{v}_c \qquad \forall\mathbf{v}_c.$$

Let $\mathbf{M}_c^1 = \frac{1}{2}\mathrm{Tr}\,(\mathbf{M}_c^0)\left(\mathbf{I} - \mathbf{N}_c(\mathbf{N}_c^\top\mathbf{N}_c)^{-1}\mathbf{N}_c^\top\right)$, where Tr denotes the trace of a matrix, and $\mathbf{I}$ is the appropriately sized identity matrix. Note that the regularity condition 3.0.1 avoids degenerate elements, and hence ensures that the edge vectors of each cell span $\mathbb{R}^2$. This guarantees that $\mathbf{N}_c$ is a full rank matrix, and hence that the inverse of $\mathbf{N}_c^\top\mathbf{N}_c$ exists. It can be shown [Veiga et al., 2014, Ch. 4] that the solution with this choice of $\mathbf{M}_c^1$ satisfies



(a) Node-edge MFD grid discretization **(b)** MAC grid discretization

Figure 3.3: In both discretizations, velocities are stored at edge midpoints, although in (b) the normal component to each cell boundary is stored, instead of the tangential component as in (a). Pressures are stored at cell centers in (b) instead of cell nodes as in (a). Note that the only essential difference between the two discretizations is which part of the domain is considered a grid cell.

the stability condition 3.1.4. In fact, this is the canonical solution used in practice [Lipnikov et al., 2014].

In summary, the MFD method prescribes an inner product matrix $\mathbf{M}$ assembled from constituent per-cell matrices $\mathbf{M}_c$, given by

$$\begin{aligned}
\mathbf{M}_c^0 &= \mathbf{R}_c(\mathbf{R}_c^\top \mathbf{N}_c)^{-1}\mathbf{R}_c^\top \\
\mathbf{M}_c^1 &= \frac{1}{2}\mathrm{Tr}\left(\mathbf{M}_c^0\right)\left(\mathbf{I} - \mathbf{N}_c(\mathbf{N}_c^\top \mathbf{N}_c)^{-1}\mathbf{N}_c^\top\right) \\
\mathbf{M}_c &= \mathbf{M}_c^0 + \mathbf{M}_c^1
\end{aligned} \tag{3.22}$$

For further details on the derivation of this solution, we direct the reader to [Lipnikov et al., 2014] and [Veiga et al., 2014]. It is worth noting that the matrix $\mathbf{M}_c$ encodes the shape of the cell, and can be precomputed for any repeating cell shape. This means that the cost of computing $\mathbf{M}$ at every frame is incurred only for cells with irregular shapes.

Example 3.1.5. Consider a regular grid mesh as in Figure 3.3. In this example we will demonstrate how to construct the mass matrix $\mathbf{M}$ on this discretization, and show that the solution (3.22) coincides with the MAC method applied to Poisson's problem (2.4) on regular grids. This equivalence is an attractive feature because the MAC discretization is widely used to simulate fluid flows in animation [Bridson, 2015, Ch. 8]. This allows one to use the MFD method as an extension to the standard MAC grid with finite differences.

Let the width (and height) of a grid cell be Δx. Considering a single grid cell c, with edges ordered: *left*, *bottom*, *right* and *top*; we can construct matrices $\mathbf{N}_c$ and $\mathbf{R}_c$ following the recipe from (3.19):

$$\begin{array}{llll} (\mathbf{N}_c)_{left} & = (1,0) & (\mathbf{R}_c)_{left} & = -\Delta x(0,-\frac{\Delta x}{2}) \\ (\mathbf{N}_c)_{bottom} & = (0,1) & (\mathbf{R}_c)_{bottom} & = \Delta x(\frac{\Delta x}{2},0) \\ (\mathbf{N}_c)_{right} & = (1,0) & (\mathbf{R}_c)_{right} & = \Delta x(0,\frac{\Delta x}{2}) \\ (\mathbf{N}_c)_{top} & = (0,1) & (\mathbf{R}_c)_{top} & = -\Delta x(-\frac{\Delta x}{2},0) \end{array}$$

This gives us local matrices

$$\mathbf{N}_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \text{ and } \mathbf{R}_c = \frac{\Delta x^2}{2} \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Then computing the matrix in equation 3.22 gives us

$$\mathbf{M}_c = \frac{\Delta x^2}{2} \mathbf{I}_4,$$

where $\mathbf{I}_4$ is the 4×4 identity matrix. Combining all the local matrices, we can see that $\mathbf{M}$ is diagonal with

$$\mathbf{M}_{e,e} = \begin{cases} \Delta x^2 & \text{if } e \text{ is an internal edge} \\ \frac{\Delta x^2}{2} & \text{if } e \text{ is a boundary edge} \end{cases}$$

Ignoring boundary conditions, we can observe how pressure is updated on the grid. Consider a pressure sample $p_{i,j}$ on a grid node at row i and column j. We will denote the velocity samples left, right, above and below the pressure sample in the grid by u_l, u_r, u_a and u_b respectively. Recall Poisson's problem discretized in (3.7). Notice that on a regular grid, a row in $\mathbf{G}^\top$ has two 1s for edge values to the *right* and *above*, and two -1s for edges to the *left* and *below*, ignoring the Δx^{-1} factor. Thus (3.7) gives the pressure solve:

$$\Delta t \underbrace{\tfrac{1}{\Delta x}(1,1,-1,-1)}_{\mathbf{G}^\top} \cdot \underbrace{\Delta x^2}_{\mathbf{M}} \underbrace{\tfrac{1}{\Delta x} \begin{pmatrix} p_{i+1,j} - p_{i,j} \\ p_{i,j+1} - p_{i,j} \\ p_{i,j} - p_{i-1,j} \\ p_{i,j} - p_{i,j-1} \end{pmatrix}}_{\mathbf{Gp}} = \underbrace{\tfrac{1}{\Delta x}(1,1,-1,-1)}_{\mathbf{G}^\top} \cdot \underbrace{\Delta x^2}_{\mathbf{M}} \underbrace{\begin{pmatrix} u_r \\ u_a \\ u_l \\ u_b \end{pmatrix}}_{\mathbf{u}^{**}}.$$

We can reduce this further to

$$\tfrac{\Delta t}{\Delta x}(4p_{i,j} - p_{i+1,j} - p_{i,j+1} - p_{i-1,j} - p_{i,j-1}) = u_b + u_l - u_a - u_r,$$

which is precisely the finite difference 5-point stencil for the Poisson problem on the MAC grid (see the book by Bridson [2015] for a neat derivation of this stencil on a MAC grid).

To emphasize the simplicity of building the inner product matrix $\mathbf{M}$ and to summarize this subsection, we present the following algorithm:

Algorithm 2 BUILD-MFD-MASS-MATRIX

for each cell c **do**
  Compute cell centroid (x_c, y_c)
  for each edge e **do**
    Let $\hat{n}$ be the outward facing normal
    Compute midpoint (x_e, y_e) and length ℓ_e of e.
    Let $\alpha_e = \hat{\tau}_e \times \hat{n}$ (scalar)
    Compute $\mathbf{r}_e = \alpha_e \ell_e \, (y_c - y_e, \, x_e - x_c)$
  end for
  Assemble $\mathbf{R}_c = [\mathbf{r}_e^\top]$ and $\mathbf{N}_c = [\hat{\tau}_e^\top]$ row-by-row
  Let $\mathbf{M}_c^0 = \mathbf{R}_c (\mathbf{R}_c^\top \mathbf{N}_c)^{-1} \mathbf{R}_c^\top$
  Let $\mathbf{M}_c^1 = \frac{1}{2} \mathrm{Tr}\,(\mathbf{M}_c^0) \left(\mathbf{I} - \mathbf{N}_c (\mathbf{N}_c^\top \mathbf{N}_c)^{-1} \mathbf{N}_c^\top \right)$
  Compute $\mathbf{M}_c = \mathbf{M}^0 + \mathbf{M}^1$
end for
return $\mathbf{M}_E = \sum_c \mathbf{M}_c$.

3.2 Boundary Conditions

In this section, we will describe how to fix boundary conditions on our discretization of Poisson's problem applied to both forms: (3.7) and (3.9).

It is most straightforward to fix a Dirichlet boundary condition, $p = f^D$, on surface nodes of the computational mesh, where pressure samples live. Assume that our pressure samples are ordered such that pressures subject to Dirichlet boundary conditions come after all other pressure samples:

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}_\circ \\ \mathbf{p}_\bullet \end{pmatrix},$$

where $\mathbf{p}_\bullet$ samples the function f^D at some surface nodes, so $\mathbf{p}_\bullet$ is known. In the same fashion, we order the gradient operator:

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_\circ & \mathbf{G}_\bullet \end{pmatrix}.$$

Then we can write (3.7) as

$$\begin{aligned}\Delta t\tilde{\mathbf{D}}\begin{pmatrix}\mathbf{G}_\circ & \mathbf{G}_\bullet\end{pmatrix}\begin{pmatrix}\mathbf{p}_\circ\\ \mathbf{p}_\bullet\end{pmatrix} &= \tilde{\mathbf{D}}\mathbf{u}^{**}\\ \Delta t\tilde{\mathbf{D}}\mathbf{G}_\circ\mathbf{p}_\circ &= \tilde{\mathbf{D}}\mathbf{u}^{**} - \Delta t\tilde{\mathbf{D}}\mathbf{G}_\bullet\mathbf{p}_\bullet,\end{aligned} \tag{3.23}$$

where $\tilde{\mathbf{D}} = \mathbf{G}^\top\mathbf{M}$ as before. Note that (3.23) is an overdetermined system, however (3.7) gives a unique solution so we can safely use only the symmetric part of (3.23), namely

$$\Delta t\tilde{\mathbf{D}}_\circ\mathbf{G}_\circ\mathbf{p}_\circ = \tilde{\mathbf{D}}_\circ\mathbf{u}^{**} - \Delta t\tilde{\mathbf{D}}_\circ\mathbf{G}_\bullet\mathbf{p}_\bullet, \tag{3.24}$$

where $\tilde{\mathbf{D}}_\circ = \mathbf{G}_\circ^\top\mathbf{M}$. Thus it suffices to solve for unknown pressures $\mathbf{p}_\circ$ in this modified system, in order to recover the full vector $\mathbf{p}$. Dirichlet conditions can likewise be enforced in (3.9). More precisely, we can move known pressure values to the right-hand-side in (3.9) to form the following system:

$$\begin{pmatrix}\mathbf{M}_E & \tilde{\mathbf{D}}_\circ^\top\\ \tilde{\mathbf{D}}_\circ & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{u}\\ \lambda_\circ\end{pmatrix} = \begin{pmatrix}\mathbf{M}_E(\mathbf{u}^{**} - \mathbf{G}_\bullet\lambda_\bullet)\\ \mathbf{0}\end{pmatrix}. \tag{3.25}$$

Note that this system is still symmetric.

Enforcing Neumann boundary conditions requires more work. In fact, we must refer to the weak formulation of the MFD discretization from Appendix A to determine how to enforce them. First we need to introduce some notation to describe the computational mesh on the surface.

Notation 3.2.1. Let N_{SV} denote the number of vertices on the surface of the discrete mesh. Then group all surface vertices $\vec{x}_i = (x_i, y_i)$ into one vector $\mathbf{x} \in \mathbb{R}^{2N_{SV}}$, where $\mathbf{x}_{2i} = x_i$ and $\mathbf{x}_{2i+1} = y_i$. In the same manner let $\mathbf{v} \in \mathbb{R}^{2N_{SV}}$ denote the vector of all surface velocity vectors $\vec{v}_i = (u_i, v_i)$ at surface vertices. Furthermore, we assume that the vertices are ordered in a clockwise direction where vertex at index i is preceded by vertex $i-1$ and succeeded by a vertex $i+1$. To keep our exposition simple, we assume a single closed loop in our discretization with identity $\vec{x}_{N_{SV}+1} = \vec{x}_1$ on the surface, however this can easily be extended to any number of closed loops representing individual separated bodies of liquid. This allows us to denote each surface edge connecting vertices i and $i+1$ by $e_{i,i+1}$, and its corresponding edge vector by $\vec{e}_{i,i+1} = \vec{x}_{i+1} - \vec{x}_i$ where its length is given by $\ell_{i,i+1} = \|\vec{e}_{i,i+1}\|$ and $\|\cdot\|$ is the standard Euclidean norm. We will also use $\ell_i = (\ell_{i-1,i} + \ell_{i,i+1})/2$ to denote the length of the patch of the surface curve local to vertex i. Furthermore, for consistency, we assume that the edge tangent unit vectors $\hat{\tau}_e$ introduced in Section 3.1 are oriented along the surface vertices in the same clockwise direction. This

allows a shorthand $\hat{\tau}_{i,i+1} = \hat{\tau}_{e_{i,i+1}}$. Finally, we let $\hat{n}_{i,i+1}$ denote the outward normal of edge $e_{i,i+1}$ and let $\hat{n}_i = (\hat{n}_{i-1,i} + \hat{n}_{i,i+1})/\|\hat{n}_{i-1,i} + \hat{n}_{i,i+1}\|$ be the vertex normal. We will use this notation below as well as in the formulation of surface tension.

The weak formulation from Appendix A prescribes a modification of (3.7) to be

$$\Delta t \tilde{\mathbf{D}} \mathbf{G} \mathbf{p} = \tilde{\mathbf{D}} \mathbf{u}^{**} - \mathbf{F} \mathbf{v} \tag{3.26}$$

where $\mathbf{F}$ is an $\mathbb{R}^{N_V} \times \mathbb{R}^{2N_{SV}}$ flux matrix defined for each surface vertex i subject to Neumann boundary conditions. Define an $\mathbb{R}^{N_{SV}} \times \mathbb{R}^{2N_{SV}}$ edge length matrix $\mathbf{L}$, where

$$\begin{aligned} \mathbf{L}_{i,2i} &= \frac{1}{2} \begin{cases} \ell_{i-1,i} & \text{if } e_{i-1,i} \text{ lies on the solid boundary} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{L}_{i,2i+1} &= \frac{1}{2} \begin{cases} \ell_{i,i+1} & \text{if } e_{i,i+1} \text{ lies on the solid boundary} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.27}$$

and an $\mathbb{R}^{2N_{SV}} \times \mathbb{R}^{2N_{SV}}$ normal projection matrix $\mathbf{N}$, defined by the 4×4 block stencil

$$\begin{pmatrix} \mathbf{N}_{2i,2i} & \mathbf{N}_{2i,2i+1} \\ \mathbf{N}_{2i+1,2i} & \mathbf{N}_{2i+1,2i+1} \end{pmatrix} = \begin{pmatrix} \hat{n}_{i-1,i}^{\top} \\ \hat{n}_{i,i+1}^{\top} \end{pmatrix} \tag{3.28}$$

for each surface vertex i. This normal matrix maps vertex velocities from their x-y coordinate representation to the normal components of neighbouring surface edges. Now the flux matrix can be written simply as

$$\mathbf{F} = \mathbf{L}\mathbf{N}.$$

This discretization gives pure Neumann boundary conditions if every vertex is subject to Neumann boundary conditions. However, this is not sufficient in general. A typical fluid will have some vertices touching a solid surface and some exposed to air. This means that we need a way to combine Neumann and Dirichlet boundary conditions. Fortunately, this is straightforward in our exposition. We simply combine (3.26) and (3.24):

$$\Delta t \tilde{\mathbf{D}}_{\circ} \mathbf{G}_{\circ} \mathbf{p}_{\circ} = \tilde{\mathbf{D}}_{\circ} \mathbf{u}^{**} - \Delta t \tilde{\mathbf{D}}_{\circ} \mathbf{G}_{\bullet} \mathbf{p}_{\bullet} - \mathbf{F} \mathbf{v}. \tag{3.29}$$

Non-homogeneous Neumann boundary conditions can also be easily enforced on (3.9) using the same formula (3.26). Using (3.8), we can relate the incompressibility part of (3.9) to (3.26), giving us a new incompressibility condition that enforces the Neumann boundary condition:

$$\tilde{\mathbf{D}} \mathbf{u} = \mathbf{F} \mathbf{v}.$$

We enforce the Dirichlet boundary condition on the contact point, since it is exposed to air. Other surface vertices that are not exposed to air, must be touching a solid boundary. This means that $\mathbf{F}$ is zero for all surface vertices subject to Dirichlet boundary conditions. Thus, we can write it as a block matrix $\mathbf{F}^\top = \begin{pmatrix}\mathbf{F}_\circ^\top & \mathbf{F}_\bullet^\top\end{pmatrix}$ where $\mathbf{F}_\bullet = \mathbf{0}$. Looking at unknown values only, we can write

$$\tilde{\mathbf{D}}_\circ \mathbf{u} = \mathbf{F}_\circ \mathbf{v}.$$

Now combining Dirichlet and Neumann boundary conditions on (3.9), gives us the system:

$$\begin{pmatrix}\mathbf{M}_E & \tilde{\mathbf{D}}_\circ^\top \\ \tilde{\mathbf{D}}_\circ & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{u} \\ \lambda_\circ\end{pmatrix} = \begin{pmatrix}\mathbf{M}_E(\mathbf{u}^{**} - \mathbf{G}_\bullet \lambda_\bullet) \\ \mathbf{F}_\circ \mathbf{v}\end{pmatrix}. \tag{3.30}$$

3.3 Alternative Discretizations

We described one possible discretization for the Poisson problem, however others are possible. For instance, Lipnikov et al. [2014] proposes to sample pressure at cell centroids (instead of mesh nodes), and velocity components normal to cell boundaries (instead of tangential). Additionally sampling pressures at cell boundaries would allow us to fix Dirichlet boundary conditions more easily. With this discretization it is natural to use a standard finite-volume type discrete divergence operator, and derive the corresponding discrete gradient operator using the MFD machinery. However, using this discretization poses significant limitations to free-surface flow as we describe in Chapter 7.

A thorough treatment of this alternative discretization, including the construction of corresponding inner product matrices, is presented by Lipnikov et al. [2014]. However, to the best of our knowledge, this chapter is the first detailed description of the node-edge based discretization in two dimensions. The extension to three dimensions of the node-edge discretization requires another inner product matrix defined in [Veiga et al., 2014, §3.4].

Chapter 4

Surface Tension

So far, we have discussed all the components of a basic simulator for incompressible free surface flow. This means that we can simulate water-like fluids, but only at large scales, where surface tension effects are not visible. This chapter will develop a method to add surface tension to our simulation in order to simulate water-like liquids at smaller scales, where surface tension effects are prominent.

Recall that pressure projection generates a divergence free velocity field, which ensures that the simulated liquid is incompressible. This requirement must hold at the end of the time step, when we move the surface of the liquid accordingly. In this chapter we will adjust our pressure projection step to accommodate surface tension forces. This gives us the flexibility to formulate a semi-implicit scheme for surface tension, but first we will introduce a basic, explicit approach.

4.1 Explicit Scheme

Consider the liquid-air interface. As described in the introduction, near this interface, water molecules tend to have a stronger attractive force between each other than between air molecules. This effect manifests in a force on the surface that tends to minimize the total surface area of the liquid. In the two dimensional case, we write the liquid-air interface as a set of k closed curves $\vec{\alpha}_k : I_k \to \mathbb{R}^2$ parametrized on some intervals $I_k \subset \mathbb{R}$, such that $\partial\Omega = \cup_k \{ \alpha_k(t) : t \in I_k \}$. Then the surface area of the liquid domain in 2D is represented

by the sum of arc-lengths of curves α_k, written as the functional

$$A[\vec{\alpha}] = \sum_k \int_{I_k} \left\| \frac{d\vec{\alpha}}{dt} \right\| dt.$$

Then, we can write the force of surface tension directly as the negative functional derivative of surface area:

$$\vec{f_{st}} = -\gamma \frac{\delta A[\vec{\alpha}]}{\delta \vec{\alpha}}, \tag{4.1}$$

where the *surface tension coefficient* γ is a physical proportionality parameter describing the strength of this force. For a water-air interface, $\gamma \approx 72\,\mathrm{mN\,m^{-1}}$, but it varies for different interfaces. We treat γ as a tuning parameter, which allows us to vary the apparent scale of the simulation. In general, conservative forces minimize an energy function, so we can also write $\vec{f_{st}} = -\frac{\delta U[\vec{\alpha}]}{\delta \vec{\alpha}}$, where $U = \gamma A$ is the surface energy.

Equation (4.1) can be directly expressed in terms of the discrete domain. Recall notation 3.2.1 describing the mesh on the surface. We can discretize the force of surface tension as

$$\mathbf{f}_{st} = -\gamma \nabla A(\mathbf{x}), \tag{4.2}$$

where the parameter of A is now a discrete vector $\mathbf{x} \in \mathbb{R}^{2N_{SV}}$ of stacked surface vertex positions and ∇ is taken with respect to this parameter. The surface area can be explicitly expressed as

$$A(\mathbf{x}) = \sum_{i=1}^{N_{SV}} \|\vec{x}_i - \vec{x}_{i+1}\|,$$

and $\vec{x}_{N_{SV}+1} = \vec{x}_1$ as before. Recall that there can be multiple connected components representing separated bodies of liquid, in which case wrapping the index $N_{SV}+1$ would be insufficient. To avoid this restriction, one can simply view the vertex $i+1$ in the formula above as the next connected vertex in the assigned clockwise order. This formulation allows us to compute the surface tension force directly from the gradient of surface area:

$$\begin{pmatrix} (\nabla A(\mathbf{x}))_{2i} \\ (\nabla A(\mathbf{x}))_{2i+1} \end{pmatrix} = \frac{\vec{x}_i - \vec{x}_{i-1}}{\|\vec{x}_i - \vec{x}_{i-1}\|} - \frac{\vec{x}_{i+1} - \vec{x}_i}{\|\vec{x}_{i+1} - \vec{x}_i\|}.$$

Then the surface tension force at vertex i can be written as

$$\vec{f_i} = \begin{pmatrix} \mathbf{f}_{2i} \\ \mathbf{f}_{2i+1} \end{pmatrix} = \gamma(\hat{\tau}_{i,i+1} - \hat{\tau}_{i-1,i}), \tag{4.3}$$

where $\hat{\tau}_{i,i+1}$ denotes the edge tangent unit vector between vertices i and $i+1$. We dropped the subscript st for brevity, and will do so for the remainder of this chapter.

In the explicit scheme, we are satisfied with computing the surface tension force from the surface in its configuration at the current time step. This approach imposes a somewhat restrictive stability condition on the time step [Brackbill et al., 1992]

$$\Delta t \propto \frac{\Delta x^{3/2}}{\sqrt{\gamma}}.$$

However, this is not too restrictive for small surface tension coefficients, so it may be sufficient for some applications in animation.

There are various ways to incorporate surface tension effects into simulation. We begin with a very straightforward one, in which surface tension is added to the pressure projection step discretized in (3.7). It involves modifying f^D in our Dirichlet boundary condition. We know from the Young-Laplace equation [de Gennes, 1985] that the jump in pressure at the liquid-air interface is proportional to curvature. In two dimensions, this is can be expressed as $\Delta p = \gamma\kappa$, where κ is the signed curvature of the boundary curve. Since we assume that air pressure is zero in our calculation, it must be that in the presence of surface tension, the pressure at the surface of the liquid must be exactly equal to this pressure jump Δp. Note that curvature is given by the second derivative of the boundary curve. On our discrete domain, at vertex i, curvature can be approximated by the discrete Laplacian:

$$(\kappa\hat{n})_i = \ell_i^{-1}(\hat{\tau}_{i,i+1} - \hat{\tau}_{i-1,i}). \tag{4.4}$$

This discretization is used for surface tension by Schroeder et al. [2012] as well as Zheng et al. [2015]. Now recall that f^D is sampled on the surface by $\mathbf{p}_\bullet$. Thus adding surface tension to (3.7) is equivalent to enforcing a non-homogeneous Dirichlet condition on each vertex i, equal to the pressure jump

$$\Delta p_i = \gamma\hat{n}_i^\top(\kappa\hat{n})_i. \tag{4.5}$$

The normal $\hat{n}_i^\top$ above is used to extract the signed mean curvature scalar κ, from the mean curvature normal vector (4.4). Using equations 4.3 and 4.4, we express the pressure boundary condition in terms of the surface tension force

$$\mathbf{p}_\bullet = \tilde{\mathbf{L}}^{-1}\tilde{\mathbf{N}}\mathbf{f}, \tag{4.6}$$

where $\tilde{\mathbf{L}}$ is an $\mathbb{R}^{N_{SV}} \times \mathbb{R}^{N_{SV}}$ diagonal length matrix defined by $\tilde{\mathbf{L}}_{i,i} = \ell_i$, and $\tilde{\mathbf{N}}$ is an $\mathbb{R}^{N_{SV}} \times \mathbb{R}^{2N_{SV}}$ normal projection matrix defined by

$$\begin{pmatrix}\tilde{\mathbf{N}}_{i,2i} & \tilde{\mathbf{N}}_{i,2i+1}\end{pmatrix} = \hat{n}_i^\top,$$

for each vertex i.

4.2 Semi-implicit Scheme

As already mentioned above, the explicit surface tension formulation bears a hefty time step restriction, especially for large surface tension forces. In this section, we will reduce the impact of this issue by adopting a semi-implicit method first introduced by Misztal et al. [2012]. We also show that this formulation is equivalent to the normal-directional semi-implicit surface tension model presented by Schroeder et al. [2012].

First, recall that the linear system (3.9) solves the constrained quadratic programming problem (3.10) as prescribed by the KKT conditions. Since surface tension force minimizes the surface energy, it is natural to add the energy term to the objective function:

$$\frac{1}{2}(\mathbf{u}-\mathbf{u}^{**})^\top \mathbf{M}(\mathbf{u}-\mathbf{u}^{**}) + U(\mathbf{x}+\Delta t\mathbf{v}). \tag{4.7}$$

In order to build an implicit scheme, we evaluate the energy at the next time step, which we get using a backwards Euler step as $\mathbf{v}$ are velocities at the next time step. Although U is non-linear, we can approximate it using a second order Taylor approximation:

$$U(\mathbf{x}+\Delta t\mathbf{v}) = U(\mathbf{x}) \;+\; \Delta t\mathbf{v}^\top \nabla U(\mathbf{x}) \;+\; \frac{1}{2}\Delta t^2 \mathbf{v}^\top \nabla\nabla U(\mathbf{x})\mathbf{v} + O(\Delta t^3). \tag{4.8}$$

Furthermore, the quadratic program minimizes over MFD edge based velocity *components* $\mathbf{u}$, so we need an interpolation scheme $\mathbf{H}$ to map $\mathbf{u}$ to *full* velocity vectors on surface vertices in $\mathbf{v}$. We will build this matrix in Section 4.2.2. For now, assuming we have this interpolation matrix, we can write the new objective function in terms of $\mathbf{u}$:

$$\frac{1}{2}\mathbf{u}^\top\mathbf{M}\mathbf{u} - \mathbf{u}^\top\mathbf{M}\mathbf{u}^{**} + \Delta t\mathbf{u}^\top\mathbf{H}^\top\nabla U + \frac{1}{2}\Delta t^2\mathbf{u}^\top\mathbf{H}^\top\nabla\nabla U\mathbf{H}\mathbf{u},$$

where the energy derivatives are evaluated at $\mathbf{x}$. Grouping first and second order terms in the new objective function together forms a new quadratic optimization problem in standard form:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{u}^\top(\mathbf{M}+\Delta t^2\mathbf{H}^\top\nabla\nabla U\mathbf{H})\mathbf{u} - \mathbf{u}^\top(\mathbf{M}\mathbf{u}^{**} - \Delta t\mathbf{H}^\top\nabla U) \\ \text{over} \quad & \mathbf{u}\in\mathbb{R}^{N_E} \\ \text{subject to} \quad & \tilde{\mathbf{D}}\mathbf{u} = \mathbf{0}, \end{aligned}$$

which can be solved with the KKT system

$$\begin{pmatrix} \mathbf{A} & \tilde{\mathbf{D}}^\top \\ \tilde{\mathbf{D}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix}, \tag{4.9}$$

where $\mathbf{A} = \mathbf{M} + \Delta t^2 \mathbf{H}^\top \nabla\nabla U \mathbf{H}$ and $\mathbf{b} = \mathbf{M}\mathbf{u}^{**} - \Delta t \mathbf{H}^\top \nabla U$ as expected. Enforcing boundary conditions on this system is identical to (3.30). However, in contrast to our formulation for explicit surface tension, we use zero Dirichlet boundary conditions since surface tension forces are applied directly in the system.

From Section 4.1, we know how to compute the energy gradient, since it is the negative of the surface tension force. Thus from equations 4.2 and 4.3 we get the expression for the energy gradient:

$$(\nabla U)_i = -\vec{f}_i = \gamma(\hat{\tau}_{i-1,i} - \hat{\tau}_{i,i+1}). \tag{4.10}$$

Thus it remains to determine the energy Hessian (i.e. $\nabla\nabla U$) and the interpolation matrix $\mathbf{H}$.

4.2.1 Building the Hessian of surface energy

For brevity, we will use the shorthand $\vec{e}_i = \vec{e}_{i,i+1} = \vec{x}_{i+1} - \vec{x}_i$ for surface edge vectors, which is unambiguous since edge vectors form a chain following the order of surface vertices. Essentially $\vec{e}_i$ is an edge starting at vertex i. In a similar fashion, we define $\hat{\tau}_i = \vec{e}_i / \|\vec{e}_i\|$. Starting from the formula for negative energy gradient in (4.3), we compute its Jacobian, giving us exactly the negative Hessian of U. Thus, for each vertex i, we can write rows $2i$ and $2i+1$ of $-\nabla\nabla U$ as

$$\begin{pmatrix} (-\nabla\nabla U)_{2i} \\ (-\nabla\nabla U)_{2i+1} \end{pmatrix} = \begin{pmatrix} \cdots & 0 & \frac{\partial f^x_{i-1}}{\partial x_{i-1}} & \frac{\partial f^x_{i-1}}{\partial y_{i-1}} & \frac{\partial f^x_i}{\partial x_i} & \frac{\partial f^x_i}{\partial y_i} & \frac{\partial f^x_{i+1}}{\partial x_{i+1}} & \frac{\partial f^x_{i+1}}{\partial y_{i+1}} & 0 & \cdots \\ \cdots & 0 & \frac{\partial f^y_{i-1}}{\partial x_{i-1}} & \frac{\partial f^y_{i-1}}{\partial y_{i-1}} & \frac{\partial f^y_i}{\partial x_i} & \frac{\partial f^y_i}{\partial y_i} & \frac{\partial f^y_{i+1}}{\partial x_{i+1}} & \frac{\partial f^y_{i+1}}{\partial y_{i+1}} & 0 & \cdots \end{pmatrix}, \tag{4.11}$$

where the terms in the first and last two rows wrap around. We do this directly using a symbolic computation package [SymPy Development Team, 2014]; Appendix B gives the code. We will write down the Hessian in a form of a recurring stencil or pattern that appears in the matrix, from which one can build the whole sparse matrix. For each surface edge $\vec{e}_i$, define two 2×2 matrices

$$E_i = \frac{\gamma}{\|\vec{e}_i\|} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{and} \qquad K_i = E_i - \gamma \frac{\hat{\tau}_i \hat{\tau}_i^\top}{\|\vec{e}_i\|}.$$

Then the Hessian of U can be built from the following stencil repeated for each vertex i:

$$\begin{pmatrix} K_i + K_{i-1} & -K_i \\ -K_i & \ddots \end{pmatrix}. \tag{4.12}$$

This energy Hessian can be interpreted as the negative derivative of the force of surface tension. Hence this formulation is equivalent to what is called the *normal-directional* implicit surface tension model in [Zheng et al., 2015], but also derived in [Schroeder et al., 2012]. This draws an equivalence between the variational formulation of semi-implicit surface tension in [Misztal et al., 2012] and the normal-directional model. For improved stability it is sometimes preferred [Zheng et al., 2015] to use a relaxed approximation for the Hessian given by the stencil

$$\begin{pmatrix} E_i + E_{i-1} & -E_i \\ -E_i & \ddots \end{pmatrix}.$$

This is referred to as the *all-directional* implicit model, since it allows the Hessian to act in the tangential direction to the surface. This is evident from the definition of K_i where a tangential projection is subtracted.

4.2.2 Building the interpolation matrix H

A keen eye will notice that using a first-order Taylor approximation for surface energy will yield an explicit surface tension scheme. The difference between this new explicit scheme, and our scheme presented in Section 4.1 is in the way we apply the force. Consider this hypothetical explicit scheme where we use the energy approximation

$$U(\mathbf{x} + \Delta t \mathbf{v}) \approx U(\mathbf{x}) \; + \; \Delta t \mathbf{v}^\top \nabla U(\mathbf{x}),$$

instead of (4.8). Following the same steps as for our implicit formulation, we form the KKT system

$$\begin{pmatrix} \mathbf{M} & \tilde{\mathbf{D}}^\top \\ \tilde{\mathbf{D}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{M}\mathbf{u}^{**} - \Delta t \mathbf{H}^\top \nabla U \\ \mathbf{0} \end{pmatrix}. \tag{4.13}$$

In Section 4.1, we applied the surface tension force as a pressure jump using a Dirichlet boundary condition. It is reasonable to expect both formulations to give the same result. Thus we compare our system (4.13) to the system (3.25), where the Dirichlet boundary conditions are set to the pressure jump caused by surface tension. Looking at the right-hand-side of both equations, it is evident that we should have

$$\Delta t \mathbf{H}^\top \nabla U = \Delta t \mathbf{M} \mathbf{G}_\bullet \mathbf{p}_\bullet. \tag{4.14}$$

Noting that $\nabla U = -\mathbf{f}$, then using equation (4.6), we can write (4.14) as

$$\mathbf{H}^\top \mathbf{f} = -\mathbf{M} \mathbf{G}_\bullet \tilde{\mathbf{L}}^{-1} \tilde{\mathbf{N}} \mathbf{f}.$$

Finally, since the interpolation should work for any surface tension force, we conclude that

$$\mathbf{H}^\top = -\mathbf{M}\mathbf{G}_\bullet\tilde{\mathbf{L}}^{-1}\tilde{\mathbf{N}}. \tag{4.15}$$

This subsection concludes the semi-implicit surface tension formulation and we have built all necessary components for free-surface liquids. In the next section, we will demonstrate stability of the semi-implicit formulation. We will also extend this model to account for solid interaction in Section 4.4.

4.3 Stability Results

The semi-implicit surface tension scheme allows us to take larger time steps in our simulations. Consider, a square of liquid floating in space, not subject to gravity, but subject to surface tension forces. Setting the surface tension coefficient and time step close to the theoretical upper bound imposed on the explicit surface tension scheme [Brackbill et al., 1992], we can observe where the explicit scheme fails. Figure 4.1 illustrates the stability advantage that the semi-implicit scheme has over the explicit surface tension scheme. Increasing the time step further will cause the simulation with explicit surface tension to fail. Further tests are compiled in the works of Schroeder et al. [2012] and Zheng et al. [2015] verifying stability of the semi-implicit scheme for different grid sizes and time steps.

4.4 Contact Angles

As we have mentioned before, surface tension is caused by an imbalance between molecular forces at the interface between two media. In the previous sections, we described how to simulate this effect for exactly two media, where the coefficient of surface tension was given as a constant γ. In a more complex scenario, we may encounter interfaces between multiple media. For instance a water droplet resting on a solid surface. In this case, there are three interfaces: liquid-air, liquid-solid and solid-air that contribute to the shape of the droplet at rest configuration. Surface tension is different at each of these interfaces, and so its force will be scaled by distinct coefficients, which we will denote by γ_{la}, γ_{ls} and γ_{sa} respectively. Accurately treating surface tension at the point where these interfaces meet (contact point) will allow us to simulate *hydrophobic* and *hydrophilic* surfaces. Hydrophilic surfaces attract water such that it spreads evenly when in contact, while hydrophobic surfaces repel water into tiny beads. This phenomenon is responsible for various visual effects that water can

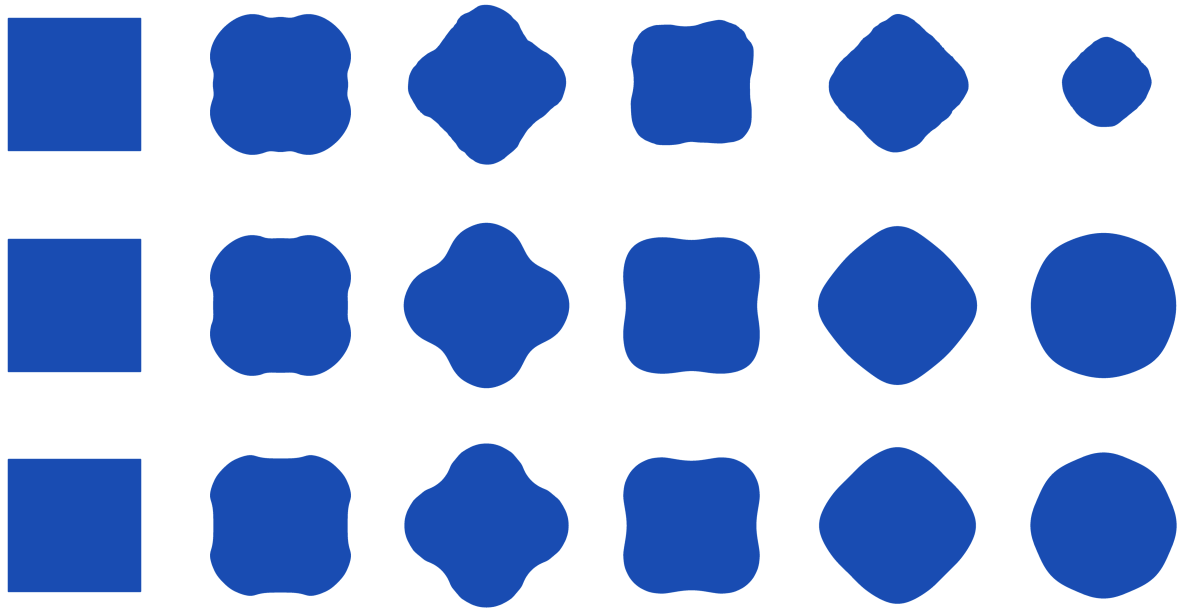


Figure 4.1: Oscillating square liquid under explicit surface tension (top row) and under semi-implicit surface tension (middle row), with surface tension coefficient $\gamma = 0.005$, grid size $\Delta x = 0.02$, grid resolution 128×128 and time step $\Delta t = 0.025\,\mathrm{s}$. Top two rows contain snapshots at times $t = 0.0$, 1.0, 3.5, 6.5, 9.0, and 21.0 seconds from left to right for the appropriate scheme. Observe the noise on the surface of the liquid with explicit surface tension (top row) due to instabilities. A side-effect of this surface noise is the aggressive volume loss in the simulation. The simulation fails in the explicit case for larger time steps. Semi-implicit surface tension is stable even for a much larger time step $\Delta t = 0.08\,\mathrm{s}$ (bottom row).

exhibit when in contact with solid surfaces as seen in Figure 4.2, which is undoubtedly important in computer graphics and animation.

"Water droplets on a geranium leave" by Philip Milne, used under CC BY-NC-ND-2.0

"Warm And Wet" by Pekka Nikrus, used under CC BY-NC-SA-2.0 / cropped from original

Figure 4.2: Examples of hydrophobic (left) and hydrophilic (right) surfaces.

Looking closely where air, liquid and solid meet, we can identify the forces acting on the contact point. It is sufficient to consider the forces of surface tension in order to identify the shape of a droplet at equilibrium.

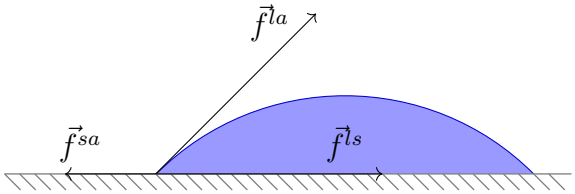
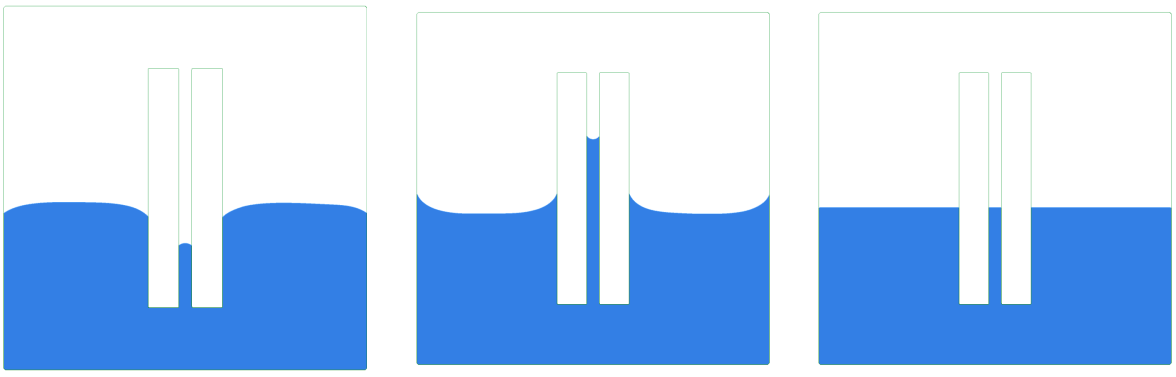


Figure 4.3: Forces of surface tension acting on the contact point of the liquid resting on a solid surface.

We will denote the surface tension forces along the liquid-air, liquid-solid, and solid-air boundaries by $\vec{f}^{la}$, $\vec{f}^{ls}$, and $\vec{f}^{sa}$ as shown in Figure 4.3, where each is proportional to the appropriate surface tension coefficient. Then assuming the solid boundary is smooth at the contact point, it is evident that the net force tangential to the solid boundary is given by $\vec{f}^{s} = \vec{f}^{ls} + \vec{f}^{sa}$, where $\vec{f}^{sa}$ points in the opposite direction of $\vec{f}^{ls}$. This makes it easy to adapt $\vec{f}^{sa}$ into our discretization, since we do not explicitly keep track of the solid-air boundary. On our discrete mesh Ω, we can extend our model of surface tension to account for these various forces. At each vertex i on the liquid-air or liquid-solid boundaries, we simply use the original formula (4.3), with appropriate coefficients:

$$\vec{f}_i = \gamma_{la}(\hat{\tau}_i - \hat{\tau}_{i-1}) \quad \text{for a liquid-air vertex}$$
$$\vec{f}_i = \gamma_{ls}(\hat{\tau}_i - \hat{\tau}_{i-1}) \quad \text{for a liquid-solid vertex.}$$



(a) $\gamma_{ls} = 0.0014, \gamma_{sa} = 0.0005$ **(b)** $\gamma_{ls} = 0.0005, \gamma_{sa} = 0.0014$ **(c)** $\gamma_{ls} = \gamma_{sa}$

Figure 4.4: Capillary action where $\gamma_{la} = 0.001$ in all 3 simulations. (a) shows a *hydrophobic* solid surface with an expected contact angle $\theta_s = 154°$. (b) shows a *hydrophilic* solid surface with $\theta_s = 26°$. (c) shows a surface that is neither hydrophobic nor hydrophilic with $\theta_s = 90°$.

Note that (4.3) is essentially a sum of two forces along each of the tangential directions away from the vertex. This is precisely the viewpoint taken in [Schroeder et al., 2012], where the forces are considered on a per-segment basis. Thus the force on each vertex i can be written as the sum of forces

$$\vec{f}_i = \vec{f}_{i,i+1} + \vec{f}_{i,i-1}$$

where $\vec{f}_{i,j}$ is the surface tension force of vertex j pulling on vertex i. Thus $\vec{f}_{i,i-1} = -\alpha\hat{\tau}_{i-1}$ and $\vec{f}_{i,i-1} = \beta\hat{\tau}_i$, where α and β are some potentially distinct surface tension coefficients. Now for the purpose of exposition, we assume that vertex $i+1$ is on a liquid-solid interface and vertex $i-1$ is on the liquid-air interface[1]. Then we can describe the surface tension at the contact vertex as

$$\begin{aligned} \vec{f}_i &= \vec{f}^{s}_{i,i+1} + \vec{f}^{la}_{i,i-1} \\ &= (\gamma_{ls} - \gamma_{sa})\hat{\tau}_i - \gamma_{la}\hat{\tau}_{i-1}, \end{aligned} \tag{4.16}$$

where we have incorporated forces $\vec{f}^{sa}$ and $\vec{f}^{ls}$ into $\vec{f}^{s}$ without an explicit representation for the solid-air boundary.

Now that we can express the force of surface tension at every vertex, we can directly use it in our explicit model for surface tension presented in Section 4.1. Furthermore, looking at (4.11) we can also easily find the new Hessian of U without having to recompute

[1]For the reverse scenario we simply swap the coefficients of the tangent vectors in (4.16).

the derivatives. Redefine E_i and K_i with $\gamma = \gamma_{la}$ for each liquid-air edge, and with $\gamma = \gamma_{ls} - \gamma_{sa}$ for each liquid-solid edge. Using the same stencils for the normal-directional and all-directional implicit models from Section 4.2.1, we get a semi-implicit surface tension model capable of simulating hydrophobic and hydrophilic surfaces.

For the purposes of visual simulation, it is rather inconvenient to have to manually choose all surface tension coefficients, and up to this point it is unclear how to pick these coefficients to achieve a specific behaviour for liquids in contact with solids. Luckily, there is a convenient relationship between these three coefficients at equilibrium given by Young's relation in terms of geometric contact angle θ_s between the liquid and solid surfaces near the contact point [de Gennes, 1985]. Young's relation is given by

$$\gamma_{sa} - \gamma_{ls} - \gamma_{la}\cos(\theta_s) = 0. \tag{4.17}$$

This relationship characterizes the spreading and beading of water on hydrophilic and hydrophobic surfaces respectively as illustrated in Figure 4.5. Equation 4.17 simplifies the control of surface tension effects in our model. More specifically, all occurrences of $\gamma_{ls} - \gamma_{sa}$ above, can be replaced by $-\gamma_{la}\cos(\theta_s)$, leaving us with just two parameters: γ_{la}, which controls the strength of surface tension at the liquid-air interface, and θ_s, which controls whether a solid surface is hydrophobic ($\theta_s > 90°$) or hydrophilic ($\theta_s < 90°$). Figure 4.4 demonstrates capillary action on hydrophobic and hydrophilic surfaces.

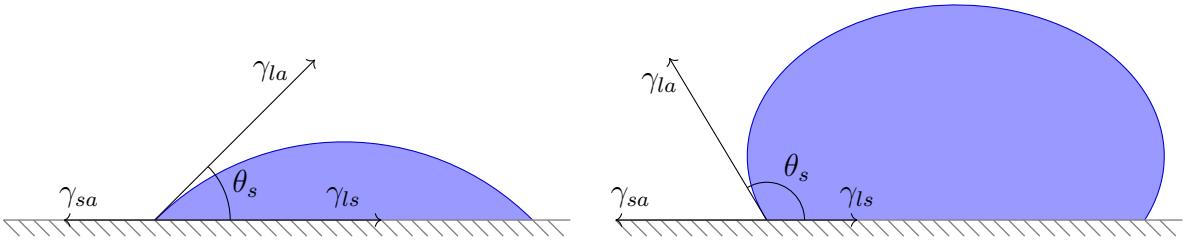


Figure 4.5: Young's relation on hydrophilic (left) and hydrophobic (right) surfaces.

4.4.1 Boundary Conditions at the Contact Point

We have identified the forces responsible for generating various contact angles. It remains to ensure that these forces are applied correctly in our model. Recall the pressure boundary condition defined in (4.6). This definition of pressure jump also defines our interpolation $\mathbf{H}$ as seen in (4.14), which ultimately defines how we apply surface tension forces in our

semi-implicit formulation. Looking at equations 4.4 and 4.5 once again more closely, we have the relationship:

$$\Delta p_i = \gamma \ell_i^{-1} \hat{n}_i^\top (\hat{\tau}_i - \hat{\tau}_{i-1}),$$

where $\hat{n}_i$ is a uniquely (in 2D) defined normal that projects the curvature vector $(\kappa \hat{n})_i$ onto the normal to extract the signed curvature value. While originally, $\hat{n}_i$ was a normal approximated at a surface vertex, now we can compute it exactly since we know the desired contact angle forming a tangent to the surface. Hence, we can compute the normal from the adjacent liquid-solid segment, say $\hat{\tau}_i$ as before, giving

$$\hat{n}_i = \hat{\tau}_i \begin{pmatrix} \sin(\theta_s - \pi) & -\cos(\theta_s - \pi) \\ \cos(\theta_s - \pi) & \sin(\theta_s - \pi) \end{pmatrix},$$

where the matrix gives a counter-clockwise rotation by $\frac{3\pi}{2} - \theta_s$ radians. Of course if $\hat{\tau}_{i-1}$ was the liquid-solid segment, the same rotation applies but in a clockwise direction, and if the vertices were ordered in a counter-clockwise order, the rotations are swapped.

Our goal now is to incorporate the new surface tension force in equation 4.16 to generate the pressure jump condition that vanishes when the desired contact angle is achieved. By rearranging terms, we can directly replace the original surface tension force (4.3) part with (4.16) to get

$$\Delta p_i = \ell_i^{-1} \hat{n}_i^\top \vec{f}_i,$$

however, this gives a non-vanishing pressure jump at equilibrium as can be seen in Figure 4.6 where the force projected onto the curvature normal $\vec{f}_i^{\,p}$, maintains a normal component to the solid surface.

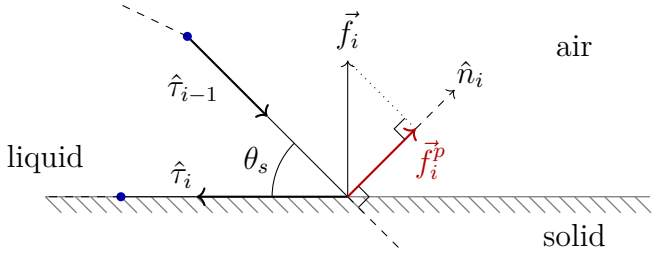


Figure 4.6: Surface tension force $\vec{f}_i$ on the contact point at equilibrium, does not vanish when projected onto the normal $\hat{n}_i$.

We ameliorate this effect by projecting the curvature normal $\hat{n}_i$ directly onto the solid surface before projecting $\vec{f}_i$. The projected normal is always given by $\hat{n}_i^p = (\hat{n}_i^\top \hat{\tau})\hat{\tau}$, where

$\hat{\tau}$ is the liquid-solid segment $\hat{\tau}_i$ or $\hat{\tau}_{i-1}$. This modification achieves the desirable result as illustrated in Figure 4.7.

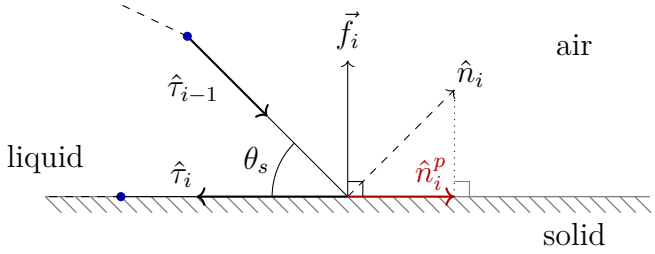


Figure 4.7: Surface tension force $\vec{f}_i$ on the contact point at equilibrium, vanishes when projected onto $\hat{n}_i^p$.

Finally, it remains to substitute the original approximated surface normal for our projected normal $\hat{n}_i^p$ at contact points, when building the interpolation matrix from (4.15). This adapts the semi-implicit surface tension formulation to handle contact angles.

Chapter 5

Mesh Generation

Up to now, we have built the tools to handle surface tension effects and simulate contact angles, but we have not discussed precisely how to represent and evolve the liquid in our simulations. In this chapter, we will describe the approach we take to generating a mesh suitable for fluid simulation. Then we will discuss the various computational stability issues that can arise and how to fix them.

In Section 6.2, we mentioned that the computational mesh consists of a regular grid on the interior, and cut cells on the surface. Following the algorithm depicted in Figure 5.1, we will make this precise. First, consider a surface mesh representing the position of the liquid surface after its vertices have been advected from the previous time step (or the initial configuration), suspended in a regular grid. This mesh, made up of connected segments, is first intersected with the background grid. Then a marching-squares-like algorithm [Müller, 2009] generates a segment per grid cell that produces a cut cell near the surface of the mesh. The only deviation from the marching squares algorithm is that we use explicit grid intersections to generate cut-cells, instead of having to interpolate an implicit function to find these intersections. In effect, avoiding interpolation helps prevent numerical smoothing artifacts, which would contribute to volume and feature loss in our simulation. The resulting mesh consists of these newly generated cut cells along with all grid cells interior to our liquid surface.

This approach requires us to store explicit mesh geometry only for the surface cells, since on the interior all cells are square. As previously mentioned, the matrix $\mathbf{M}$ is diagonal for interior cells and can be precomputed once at the start. For this reason, the MFD method also lends itself well to unrestricted quad-tree optimizations, without the need for auxiliary schemes to construct well behaved discrete Laplacians [Losasso et al., 2005].

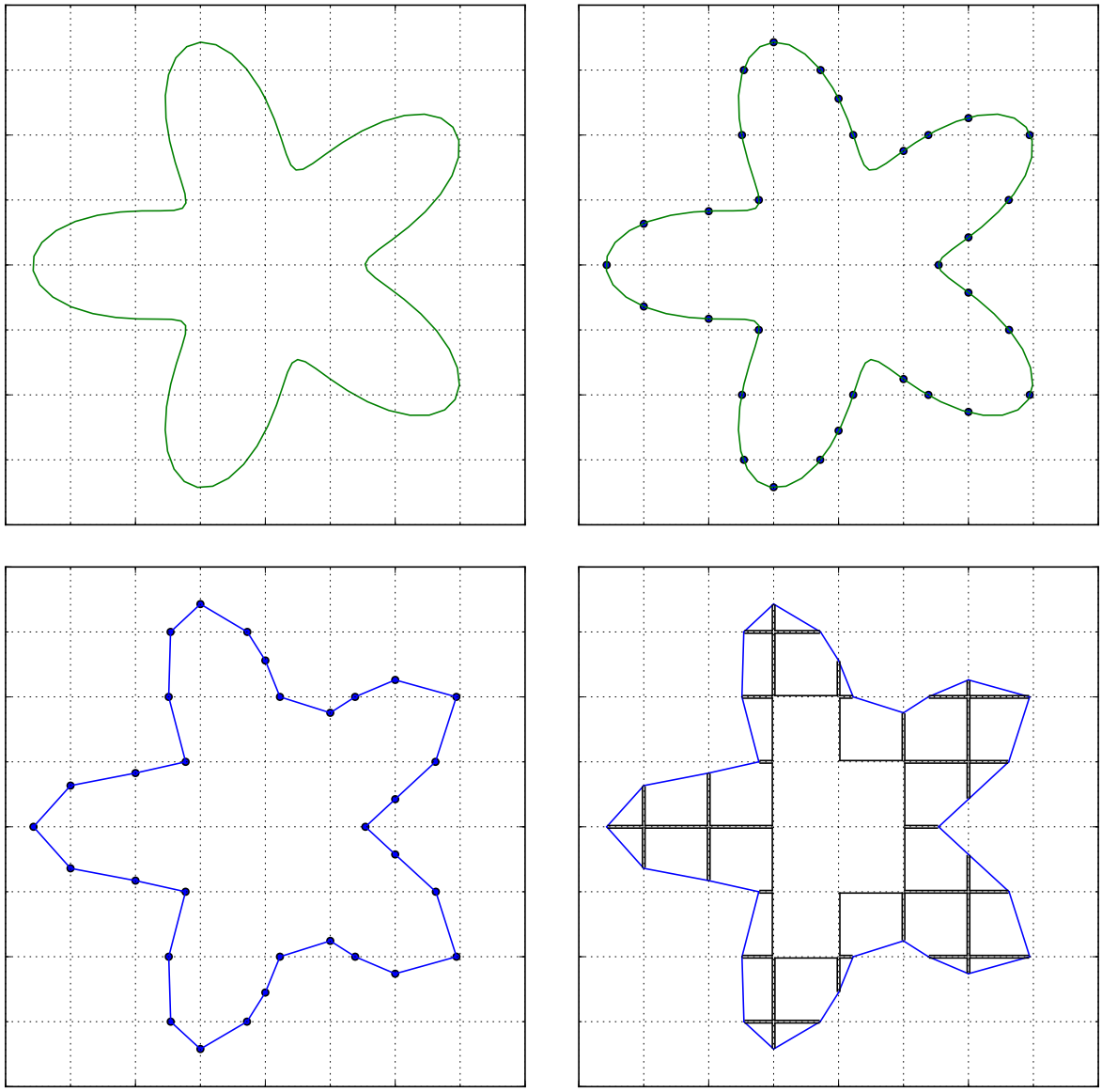


Figure 5.1: Generating a cut-cell mesh. Starting with an initial surface mesh (top-left), we compute grid intersections (top-right), construct marching squares segments (bottom-left), and finally cut out cells near the surface (bottom-right) outline with black lines. The remaining square cells on the interior of the liquid are referred to as interior cells.

More precisely, one can precompute $\mathbf{M}_c$ for a variety of different quad-tree cells, and reuse them throughout the tree as needed taking advantage of various symmetries. Of course unrestricted quad-trees can have arbitrarily varying gradings between neighbouring cells, so for some cells, $\mathbf{M}_c$ would need to be computed on demand. For more details on solving the Poisson problem on quad-trees including a summary of competitive methods see the work by Batty [2015].

Unfortunately, this approach to creating meshes can produce small angles, when the initial surface is close to a grid node. As a result, the pressure projection step can suffer from instabilities. In particular small angles cause a restriction on the mesh regularity condition 3.0.1. It is not hard to see that a small angle in a cell can restrict the ratio in (3.1) for admissible triangulations under refinement. Consider a triangle shaped cut-cell with a small angle θ produced as a result of our meshing methodology. Any subdivision of such a triangle is guaranteed to contain a polygon retaining the angle θ. Since the stability condition 3.1.4 (and hence the spectral bound on $\mathbf{M}$) depends on this ratio bound, the conditioning of the matrix $\mathbf{M}$ also suffers. Fortunately, a simple remedy for this problem is to collapse small edges near the surface of the mesh since the MFD method is not restricted to particular polygonal shapes. This method is discussed in detail in Section 5.3.

Another shortcoming of our approach comes from the surface tension formulation. In particular looking closely at the interpolation matrix $\mathbf{H}$ defined in (4.15), we find its dependence on the inverse of surface edge lengths encoded in $\tilde{\mathbf{L}}^{-1}$. In the presence of the edges with lengths significantly smaller than other edges on the surface, the diagonal matrix $\tilde{\mathbf{L}}^{-1}$ can become nearly singular. This problem motivates a more complex solution, which involves a relaxation step on the surface mesh. Section 5.4 covers the details of relaxing vertices on the surface to form a more uniform edge length distribution.

Our approach to preserving the contact point using clipping (see Section 5.2), may produce poor quality cells that can also contribute to poor conditioning of $\mathbf{M}$. In particular, some small angles can arise at the contact point, which cannot be removed by collapsing small edges. We noticed that merging such cells with their neighbours can significantly improve stability.

Of course numerical stability issues can arise when edge lengths approach the limits of available floating point precisions. In this case, we simply merge vertices together. This doesn't produce significant visual artifacts because this type of operation happens rarely and at scales on the order of machine epsilon, so not typically visible.

Before we proceed to dealing with these issues, we first need to describe how the fluid moves from one time step to the next, and how liquid and solid surfaces are coupled.

5.1 Moving the Surface Mesh

So far we have subtly mentioned the need to move a surface mesh to simulate the motion of the liquid, but we have not precisely defined what we mean. This is indeed a very important part of simulating liquids, because the motion of the surface is what characterizes how we perceive the liquid. We will discuss two methods to advect the surface mesh through the grid and compare the two qualitatively.

Recall that after our pressure projection step, we get a discretization of the divergence free velocity field $\vec{u}(t + \Delta t)$ for the next time step, where Δt is the time step and t is the current time. This naturally allows us to use a stable backward Euler step given the position of the surface $\vec{x}(t)$ at the current time step to get the vertex positions for the next time step $\vec{x}(t + \Delta t)$ as

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \Delta t \vec{u}(t + \Delta t). \tag{5.1}$$

Now looking at the precise discretizations of our velocity and position variables, we note that velocities $\mathbf{u} \in \mathbb{R}^{N_E}$ are stored on the surface edge midpoints, but $\mathbf{x} \in \mathbb{R}^{N_S V}$ samples the liquid surface at surface vertices. Thus we cannot simply apply the formula (5.1) directly on our discretization. An attentive reader may notice that in our surface tension discretization we constructed an interpolation operator $\mathbf{H}$ that maps tangential edge velocity components in $\mathbf{u}$ to full vertex based velocities $\mathbf{v} \in \mathbb{R}^{2N_S V}$ on the surface. Indeed, this allows us to apply the backwards Euler step as

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \mathbf{H} \mathbf{u}^{t+\Delta t}.$$

This is the simplest method for moving the surface and it stays consistent with the MFD methodology, thus we will refer to it as *MFD-surface-advection*. However, as we will see later, this approach has its shortcomings.

Alternatively, we may build a separate interpolation operator with a focus on velocities, as opposed to forces, which were the focus when building $\mathbf{H}^\top$. First, consider a vertex at the surface of the mesh $\vec{x}_i$. In our discretization $\vec{x}_i$ is incident to some edges, which we define by the set $\mathcal{E}_i = \{\, e \;:\; e \text{ is incident to } i \,\}$. Let $\mathbf{u}_i \in \mathbb{R}^{|\mathcal{E}_i|}$ denote the vector $\mathbf{u}$ restricted to edges incident to vertex i, and let $\mathbf{v}_i = (\mathbf{v}_{2i}, \mathbf{v}_{2i+1})^\top$ be a shorthand for the target velocity at vertex i. Then we can write the relationship between $\mathbf{u}_i$ and $\mathbf{v}_i$ as a least-squares system $\mathbf{u}_i = \tilde{\mathbf{H}}_i \mathbf{v}_i$, where each row of $\tilde{\mathbf{H}}_i$ representing an incident edge is given by $(\tilde{\mathbf{H}}_i)_e = \hat{\tau}_e^\top$. Solving this small least-squares system at every vertex and combining the resulting velocities into one vector will give us $\mathbf{v}^{t+\Delta t} \in \mathbb{R}^{2N_S V}$ as desired. Sometimes the only tangent vectors incident to a vertex i are parallel. In this case, we add another row into the least-squares matrix $\tilde{\mathbf{H}}_i$ with $(0, 1)$ or $(1, 0)$ to resolve the missing horizontal or

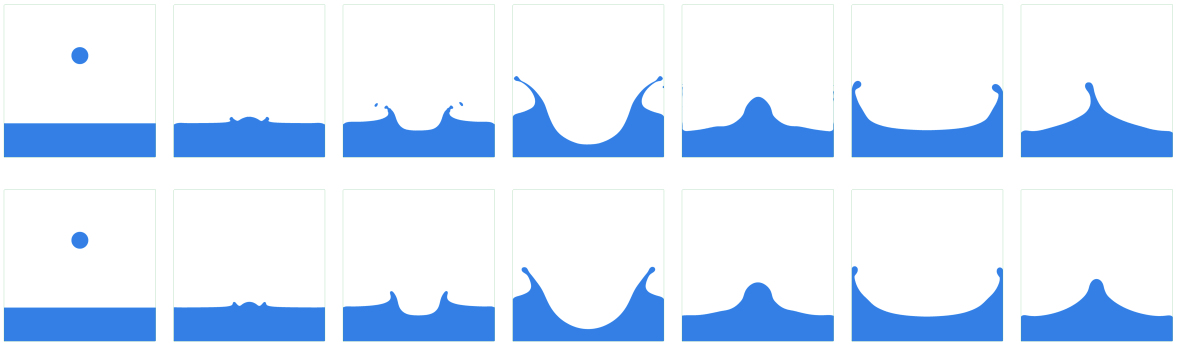


Figure 5.3: MFD-surface-advection (top row) is compared to LS-surface-advection (bottom row) to see the visual difference in the full "falling droplet" simulation.

vertical velocity component respectively. On the right hand side, we add a velocity sample from the grid to $\mathbf{u}_i$. Finally, this allows us to apply the backwards Euler step once again as

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \mathbf{v}^{t+\Delta t}.$$

Naturally we will refer to this method as *LS-surface-advection*.

Notice, from the appearance of $\mathbf{N}$ in the definition of $\mathbf{H}$, that we can only build vertex velocities $\mathbf{v}$ normal to the surface. This is the main shortcoming of the MFD-surface-advection approach. In contrast, the LS-surface-advection approach resolves the full velocity at a surface vertex. To illustrate the difference, we move a circular droplet through the grid with a constant downward velocity. After some time, the droplet deforms from its original circular state as shown in Figure 5.2. As we can see the droplet deforms significantly from its original shape under MFD-surface-advection. On the flip side, the MFD-surface-advection method produces a more dynamic simulation, while the LS-surface-advection naturally smooths the velocity field and hence dampens the simulation. Consider Figure 5.3 to see that the MFD-surface-advection scheme produces finer details on the liquid surface like droplet pinch-off in the third frame.

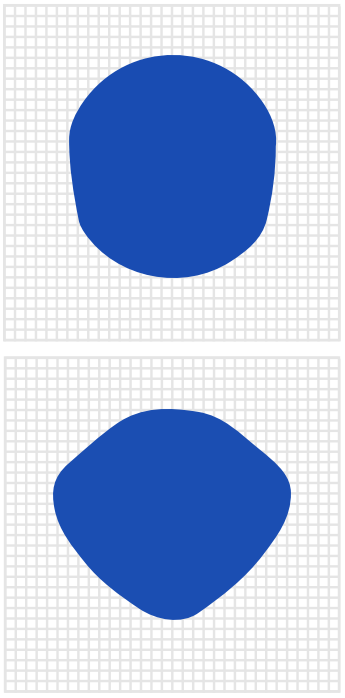


Figure 5.2: Droplet deformed under LS-surface-advection (top) and MFD-surface-advection (bottom).

Finally, after moving surface vertices, we are left with a surface mesh composed of connected line segments. From this

point, we can create a simulation cut-cell mesh as we have seen in Figure 5.1. This concludes all the fundamental steps of our simulator. What follows are methods we chose to improve the stability and quality of the simulation. However, it is important to note that without these improvements, the simulation is unstable. It may be possible to find alternatives, but we found the combination of the following meshing techniques to be both simple and robust.

5.2 Liquid-Solid Mesh Generation

So far we have talked about contact points between liquid-air and liquid-solid boundaries, but we have not mentioned how these are formed. In this section, we will describe this process in detail. Our goal is to create a method to track contact points accurately to preserve the visual aesthetics of surface tension at the point of contact.

To start we need two initial meshes: one for the liquid surface L and one for the solid surface S. First, we execute the algorithm in Figure 5.1 on L to get a set of convex cells L'. Then we can execute the first three steps of the same algorithm on S giving us a new solid segment mesh $\tilde{S}$ aligned with the grid. Alternatively, starting with an implicit function ϕ whose zero level set approximates S, we can run a standard marching squares algorithm on $\phi = 0$ to build $\tilde{S}$ instead. In any case, $\tilde{S}$ is a grid aligned mesh with one surface segment per grid cell. Then for every liquid cell $c \in L'$, containing a segment $s \in \tilde{S}$, we clip c against s generating a new clipped cell c' with a portion of the boundary lying on the solid surface. Figure 5.4 shows the five arm flower from Figure 5.1 clipped against a circular solid cavity. It is worth noting that all cells c are convex and have mostly axis aligned boundaries, which makes clipping especially simple. In contrast, clipping arbitrary non-convex meshes is error-prone and typically requires sophisticated algorithms with complicated edge cases. This is not such a crippling issue in two dimensions, but becomes more important with 3D implementations.

This method allows us to represent the contact point within a grid cell. That is, the point need not lie on the boundary of a cell. However, at the very next time step, the surface mesh representing the liquid surface will move and our mesh generation algorithm will smooth away the contact point because the standard marching cubes algorithm is not able to represent surface details within a grid cell. To see this clearly, consider Figure 5.5, which illustrates a moving liquid boundary that gets smoothed away during the next mesh generation step. To alleviate the problem we could use an extended marching cubes approach [Kobbelt et al., 2001], which is able to preserve the contact vertex. However, to keep our implementation simple, we chose to simply project all surface vertices on the

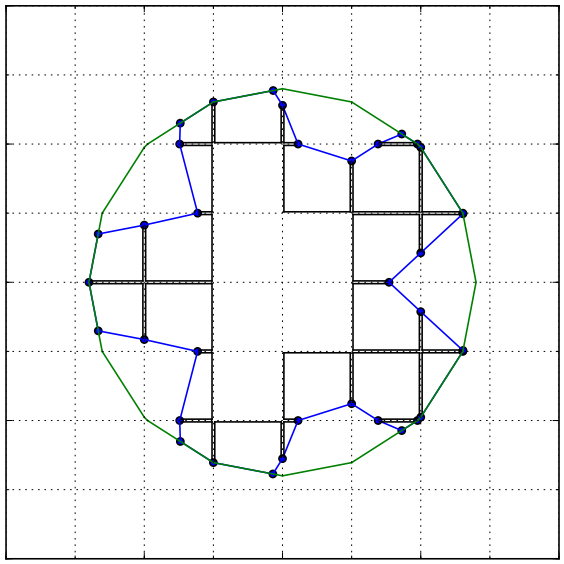


Figure 5.4: Liquid mesh of a five arm flower is clipped against a circular cavity (green).

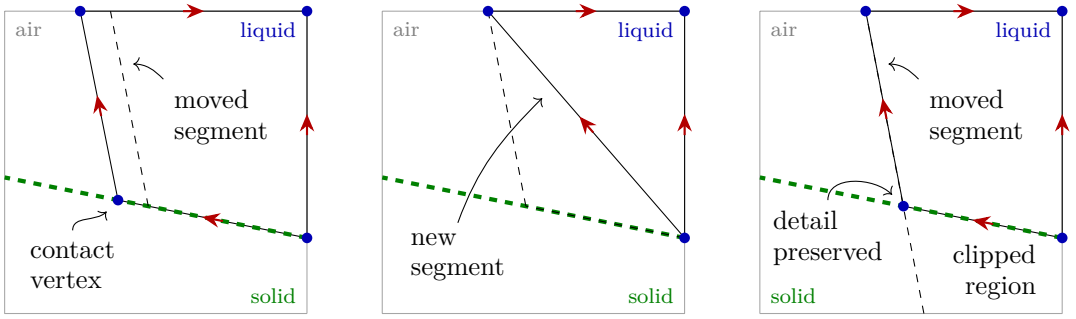


Figure 5.5: Surface with a contact vertex at time t (left) moves to the next configuration at time $t + \Delta t$ losing the contact point representation due to the limitations of marching squares (middle). This issue is resolved by moving the contact vertex into the solid such that it intersects a grid edge (before moving the liquid-air segment) and clipping (right).

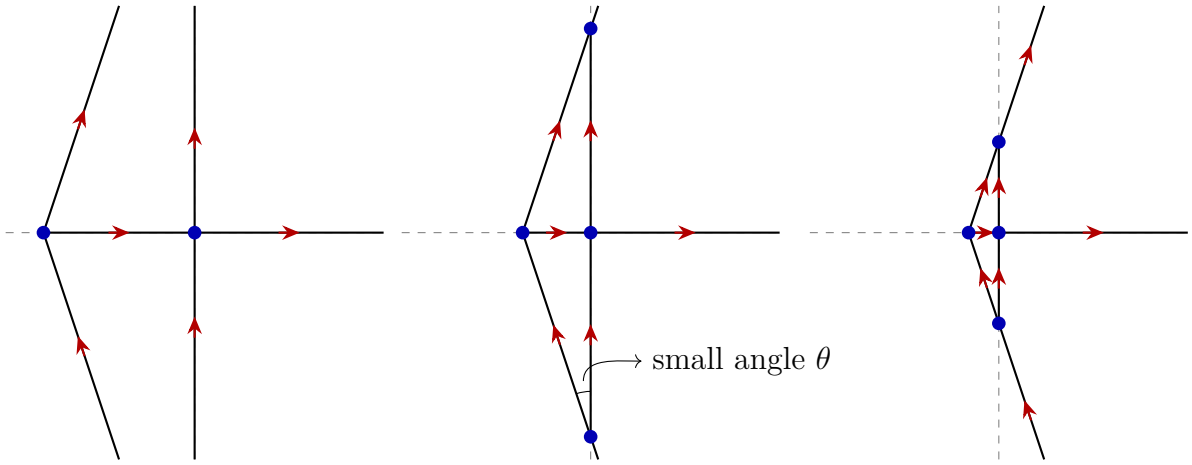


Figure 5.6: An example of a mesh moving to the right, closer to the grid node, potentially forming triangles with small angles. Grid lines are dashed in grey, while mesh edges are represented by thick black lines.

solid boundary (including the contact vertex) into the solid before generating the new simulation mesh. If we move the vertices into the solid by twice the size of a grid cell Δx, then our clipping step will guarantee to preserve the contact point as shown on the bottom of Figure 5.5. Note that we move the contact vertex in the direction parallel to the edge vector toward the interior of the solid, while other solid vertices are simply moved along the normal of the solid surface. Although this method limits the thickness of admissible solids, we can increase the grid resolution to allow for thinner solids. Furthermore, this approach prevents clipping nearly parallel segments against each other since the fluid surface at the solid boundary is moved into the solid.

5.3 Collapsing Small Edges

Recall that our simulator moves an explicit surface mesh through a regular grid and creates a cut cell interior mesh at every step. This means that it is possible to generate cells with small angles quite easily. Consider Figure 5.6, which illustrates a mesh moving closer to a grid node. This example shows pathological cell structures that can be generated with cut cell geometry. In order to avoid cells with small angles, we will simply collapse the smallest interior mesh edge near a grid node if the edge is smaller than some threshold δ. More precisely, iterating over each interior grid node, we find the smallest edge e, and if $|e| < \delta$ then we remove it by stitching the end points of e together to maintain mesh topology.

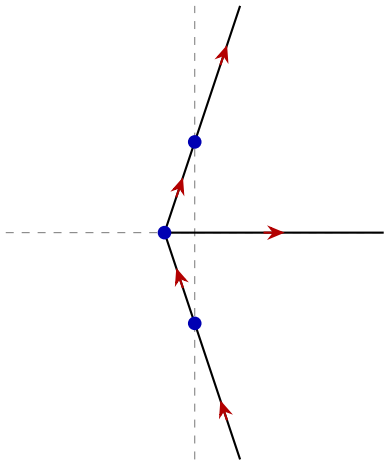


Figure 5.7: Mesh with small edges removed.

Collapsing a small edge on the interior mesh may produce degenerate cells with zero area, if for instance, the edge was part of a triangle cell, which is exactly the case in Figure 5.6. Thus an extra step is required to remove any possible resulting degenerate cells. Using this technique on cut cell geometry naturally guarantees that all interior edges in the mesh will have length greater than or equal to δ. Figure 5.7 illustrates how this method removes the leftmost interior edge arising in Figure 5.6. In addition, this method of removing small interior edges leaves the original surface mesh intact. In contrast, other remeshing techniques often seen with finite-element methods (e.g. [Clausen et al., 2013]), which make changes to the surface mesh, exhibit flickering effects on the surface. Furthermore, since the MFD method works with irregular cells, no further triangulation steps are needed, whereas many standard FEM schemes for unstructured meshes require triangular elements.

Lastly, note that the above operations of collapsing edges, forces the MFD mesh to be slightly misaligned with the background grid near the surface of the liquid. As a result, we cannot simply copy all MFD velocity components to the grid for use in velocity advection at the next time step (see Algorithm Unified-MFD for reference). Thus for those velocity components misaligned with the grid, we propose the following interpolation scheme. Note that the rest of the grid velocity components are exactly aligned with the MFD mesh, so we can simply copy them directly onto the grid. This interpolation scheme is needed prior to velocity advection, so before between lines 4 and 5 in Algorithm Unified-MFD. Note that we also need to copy/interpolate the grid velocities onto the MFD mesh before the incompressibility step on line 7 of Algorithm Unified-MFD. We can use a simple bilinear interpolation technique here, to get a velocity sample on an MFD edge that is misaligned

Algorithm 3 MFD-Grid-Interpolation

Compute a full velocity vector $\vec{v}_n$ for each MFD node n using a least squares scheme as employed in Section 5.1.
Compute a full velocity vector $\vec{v}_c$ on the centroid of every MFD surface cell c.
for each grid edge e **do**
  Let $\vec{x}_e$ be the midpoint of edge e.
  if $\vec{x}_e$ lies inside a surface MFD cell c, **then**
    Let (n, m) be an MFD edge that was aligned with e before small edge collapse.
    Let w_n, w_m and w_c be the barycentric coordinates of $\vec{x}_e$ with respect to the triangle formed by $\vec{x}_n$, $\vec{x}_m$ and $\vec{x}_c$, such that $\vec{x}_e = w_n\vec{x}_n + w_m\vec{x}_m + w_c\vec{x}_c$.
    $(v_x, v_y) \leftarrow w_n\vec{v}_n + w_m\vec{v}_m + w_c\vec{v}_c$.
    $u_e \leftarrow v_x$ if e is horizontal, or v_y if e is vertical.
    print the new grid velocity u_e.
  end if
end for

with the grid. It's important to note that this extra interpolation step between grid and MFD velocities is only required be cause we employed a semi-Lagrangian, grid-based velocity advection scheme. When using a particle based velocity advection scheme, it is sufficient to interpolate directly between particles and the MFD mesh.

5.4 Mesh Relaxation

We now shift our focus to small edges on the *surface* of the simulation mesh. It is better to avoid making explicit changes to the surface mesh as it will break temporal coherence in the mesh geometry from frame to frame. This causes spurious flickering artifacts as the mesh moves through the grid, and more importantly, it causes sudden intermittent changes in curvature estimates, and thus in surface tension forces, which can lead to instability. Consequently, we propose a simple greedy relaxation step on the surface mesh that we will only use to smooth the surface tension force applied to each surface vertex and scaled by the lengths ℓ_i^{-1}. More specifically, we will build an auxiliary surface mesh S that approximates the original mesh T, but maintains a more uniform spacing between neighbouring vertices. Then we will compute the forces scaled by ℓ_i^{-1} on S and linearly interpolate the result back to T. Thus, we need a smoothing operator $\mathbf{S}$ that will modify our interpolation matrix $\mathbf{H}$,

originally defined in (4.15), as follows:

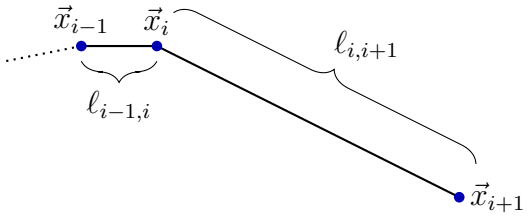
$$\mathbf{H}_S^\top = -\mathbf{M}\mathbf{G}_\bullet\tilde{\mathbf{N}}\mathbf{S}\tilde{\mathbf{L}}_2^{-1}, \tag{5.2}$$

where we swapped the order of the normal projection $\tilde{\mathbf{N}}$ and $\tilde{\mathbf{L}}_2^{-1}$, which is defined by repeating each diagonal entry of $\tilde{\mathbf{L}}^{-1}$, or more exactly by the Kronecker product

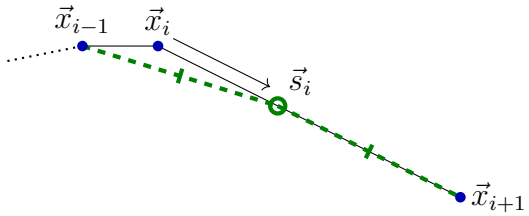
$$\tilde{\mathbf{L}}_2^{-1} = \tilde{\mathbf{L}}^{-1} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that this swapping does not change the original matrix $\mathbf{H}^\top$ since $\tilde{\mathbf{L}}^{-1}\tilde{\mathbf{N}} = \tilde{\mathbf{N}}\tilde{\mathbf{L}}_2^{-1}$. Recall that $\mathbf{H}^\top$ is applied to full force vectors on surface vertices of T. The new operator $\mathbf{H}_S^\top$ acts on force vectors on surface vertices of S instead. Thus to break down equation 5.2, surface tension forces $\mathbf{f}$ defined on the smoothed surface vertices of S are first scaled by the inverse lengths ℓ_i^{-1} by $\tilde{\mathbf{L}}_2^{-1}$. Then the result is interpolated back onto the original mesh T by $\mathbf{S}$. Finally a pressure jump is computed by projecting the result onto the curvature normal using $\tilde{\mathbf{N}}$ and the rest is unchanged from equation 4.15.

To build the mesh S, first consider a vertex i on the surface of T, which connects a small edge $e_{i-1,i}$ to a large edge $e_{i,i+1}$ as depicted in Figure 5.8a. Then move the vertex $\vec{x}_i$ to a new position $\vec{s}_i = t_i\vec{x}_i + (1-t_i)\vec{x}_{i+1}$ such that $\|\vec{s}_i - \vec{x}_{i-1}\| = \|\vec{s}_i - \vec{x}_{i+1}\|$ as shown in Figure 5.8b. If on the other hand edge $e_{i-1,i}$ is much larger than $e_{i,i+1}$, naturally we move $\vec{x}_i$ towards $\vec{x}_{i-1}$ instead. This procedure repeated for each vertex creates the mesh S with more evenly distributed edge lengths with one exception. Suppose that $\vec{x}_i$ is moved towards $\vec{x}_{i-1}$, but $\vec{x}_{i-1}$ is moved towards $\vec{x}_i$. Then the new edge between $\vec{s}_i$ and $\vec{s}_{i-1}$ can potentially become arbitrarily small. We resolve this issue by simply clamping the value of t_i between 0.6 and 1 to ensure such edges do not creep up in our relaxation algorithm.



(a) Original surface mesh T, at vertex $\vec{x}_i$, where $\ell_{i-1,i} \ll \ell_{i,i+1}$.



(b) Relaxation step moving vertex $\vec{x}_i$ to a new position $\vec{s}_i$.

Figure 5.8: Greedy relaxation step moving a vertex $\vec{x}_i$ to a position to improve the spacing between neighbouring vertices.

Algorithm Surface-Relaxation precisely describes how to compute the smoothed mesh S along with the weights t_i needed during interpolation. It remains to interpolate values computed at $\vec{s}_i$ (such as the force of surface tension) onto the original vertices $\vec{x}_i$. Since each vertex can be moved in either direction along the mesh, we cannot directly use the values t_i as weights for our interpolation. To illustrate this, consider vertices $\vec{x}_i$ and $\vec{x}_{i-1}$ moved away from each other during the relaxation step as shown in Figure 5.9a. Let v_i^S be the values at surface vertices $\vec{s}_i$ that we would like to interpolate to the vertices $\vec{x}_i$ to get values v_i^T. Note that v_i^T and v_i^S can be scalar or vector values in general (although in our case they will be force vectors). Then to help express the interpolation procedure, define two indices: $k(i)$ representing the vertex that vertex i is moving towards, and $n(i)$ representing the vertex that vertex i is moving away from as follows:

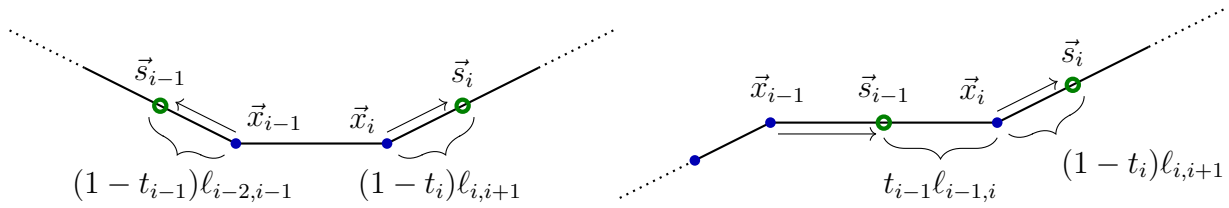
Algorithm 4 Surface-Relaxation

for each surface vertex $\vec{x}_i$ **do**
  $\vec{e} \leftarrow \vec{x}_{i+1} - \vec{x}_{i-1}$
  if $\ell_{i,i+1} > \ell_{i-1,i}$ **then**
    $t_i \leftarrow \dfrac{\|\vec{e}\|^2}{2\vec{e} \cdot \vec{e}_{i,i+1}}$
    $\vec{s}_i \leftarrow t_i \vec{x}_i + (1 - t_i)\vec{x}_{i+1}$
  else
    $t_i \leftarrow \dfrac{\|\vec{e}\|^2}{2\vec{e} \cdot \vec{e}_{i-1,i}}$
    $\vec{s}_i \leftarrow t_i \vec{x}_i + (1 - t_i)\vec{x}_{i-1}$
  end if
  print t_i and $\vec{s}_i$
end for

$$k(i) = \begin{cases} i+1 & \text{if } \ell_{i,i+1} > \ell_{i-1,i} \\ i-1 & \text{otherwise} \end{cases} \qquad n(i) = \begin{cases} i-1 & \text{if } \ell_{i,i+1} > \ell_{i-1,i} \\ i+1 & \text{otherwise.} \end{cases}$$

Then we can write the interpolation scheme as

$$v_i^T = \frac{w}{w+w'} v_i^S + \frac{w'}{w+w'} v_{n(i)}^S,$$



(a) Vertices are moved apart from each other. **(b)** Vertices are moved in the same direction.

Figure 5.9: Possible cases considered for linear interpolation weights defined in (5.3).

where the weights w and w' are defined as

$$w = \begin{cases} t_{n(i)}\ell_{n(i),i} & \text{if } k(n(i)) = i \quad \leftarrow \text{ the case in Figure 5.9b} \\ (1 - t_{n(i)})\ell_{k(n(i)),i} + \ell_{n(i),i} & \text{otherwise} \quad \leftarrow \text{ the case in Figure 5.9a} \end{cases} \tag{5.3}$$
$$w' = (1 - t_i)\ell_{k(i),i},$$

and this is the only place we use $\ell_{i+1,i} = \ell_{i,i+1}$ interchangeably. In plain words, to find the value v_i^T we simply trace along the surface toward the closest values $v_{n(i)}^S$ and $v_{k(i)}^S$ and use the arc-length along the path to compute the interpolation weights. If we instead think about the individual weighted contributions of values v_i^S to values v_i^T, it's not hard to see an extension of this algorithm to an arbitrary weighted average of the form

$$v_i^T = \frac{1}{w + \sum_{j \in N(i)} w_j}(w v_i^S + \sum_{j \in N(i)} w_j v_j^S)$$

where $N(i)$ is a set of surface vertex indices neighbouring vertex i. This generalization should be useful for extending this interpolation algorithm to three dimensions.

Finally, collecting the values v_i^T and v_i^S into vectors $\mathbf{v}^T \in \mathbb{R}^{N_{SV}}$ and $\mathbf{v}^S \in \mathbb{R}^{N_{SV}}$ respectively, allows us to write the interpolation as a matrix operator $\mathbf{v}^T = \mathbf{S}\mathbf{v}^S$, where the interpolation matrix $\mathbf{S}$ is naturally defined on each row i by

$$\begin{pmatrix} \mathbf{S}_{i,i} & \mathbf{S}_{i,n(i)} \end{pmatrix} = \begin{pmatrix} \frac{w}{w + w'} & \frac{w'}{w + w'} \end{pmatrix}. \tag{5.4}$$

This choice of relaxation conveniently preserves the number of degrees of freedom on the surface, and provides a straightforward linear interpolation scheme. It is important to note that contact vertices i connecting the liquid-air surface to the solid-liquid surface are pinned to their original locations in T, by the constraint $t_i = 1$, in order to preserve correct behaviour at the contact point.

Chapter 6

Results

A serial implementation of our method is written in C++ and all simulations were performed on a 2.3GHz Intel Core i7 machine with 16 GB memory. We use the Eigen library [Guennebaud et al., 2010] for matrix manipulations and linear solves. In particular, we used the sparse LU factorization based solver to solve (4.9). The solver incorporates all elements described in this thesis including semi-Lagrangian velocity advection, MFD pressure projection, semi-implicit surface tension, and enhanced surface tracking. In addition, we use a volume control method [Kim et al., 2007] to prevent volume loss, which can otherwise be observed in highly dynamic or lengthy simulations such as in the top row of Figure 4.1. It is not hard to see that our choice for surface tracking is particularly sensitive to volume perturbations since the surface mesh defining the simulation volume is reconstructed at every step approximating the true moved surface.

In this chapter, we present a number of examples demonstrating the capabilities of our liquid simulator. The first two sections verify the convergence of our MFD based pressure projection step subject to Dirichlet and Neumann boundary conditions. The last section provides a number of examples for qualitative verification of our simulator. All measurements use the standard SI units (e.g. meters for distance).

6.1 Numerical Results on a Regular Domain

In this section, we test the convergence of the MFD method as applied to the Poisson problem $\nabla \cdot \nabla p = \nabla \cdot \vec{u}_{in}$ on a square domain $\Omega = [-1, 1]^2$. We discretize the domain using a regular grid as in Figure 3.3a. For the following tests in this section, we use an alternative implementation using the Julia [Bezanson et al., 2012] programming language.

To test convergence with respect to grid resolution, we will compare a known pressure solution p to a computed pressure $\mathbf{p}_h$, sampled on the nodes of the grid, given an input velocity field $\vec{u}_{in}$. In addition we will compute the output velocity $\mathbf{u}_h = \mathbf{u}_{in} - \mathbf{G}\mathbf{p}_h$ field and compare it to the expected divergence free velocity field $\vec{u}_{out}$ to test for convergence in the velocity variable. Given a target analytic pressure p we compute its gradient ∇p and using the target velocity field $\vec{u}_{out}$ we compute $\vec{u}_{in}$ as

$$\vec{u}_{in} = \vec{u}_{out} + \nabla p \tag{6.1}$$

It is important to note that the results below merely verify that the MFD method is identical to the staggered grid approach to solving Poisson's equation on a regular grid. As such, we expect second order convergence in both pressure and velocity variables. The purpose of this section is to demonstrate the correct use of Dirichlet and Neumann boundary conditions, required to simulate the behaviour of a fluid at the air and solid interfaces respectively.

For reference, Figures 6.1 and 6.2 illustrate the scalar and vector functions used throughout for the following numerical tests respectively.

6.1.1 Dirichlet Boundary Conditions

Example 6.1.1. This example verifies convergence of pressure projection subject to pure homogeneous Dirichlet boundary conditions. We define the test functions for pressure p, as well as output and input velocities, $\vec{u}_{out}$ and $\vec{u}_{in}$ respectively:

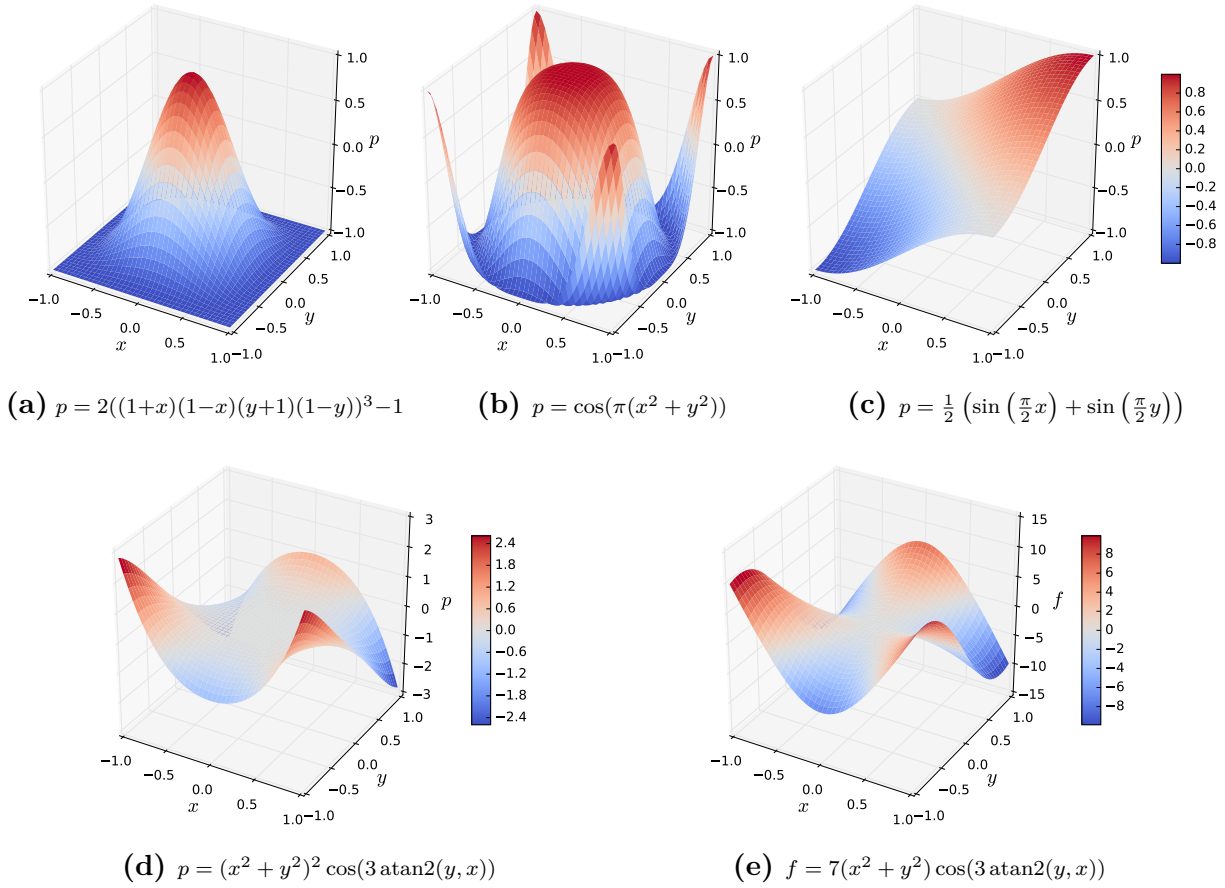
$$p = 2((1+x)(1-x)(y+1)(1-y))^3 - 1 \qquad \text{(see Figure 6.1a)}$$
$$\vec{u}_{out} = (y, -x)^\top \qquad \text{(see Figure 6.2b)}$$

where $\vec{u}_{in}$ is given as in (6.1). Table 6.1 demonstrates second order convergence of the MFD scheme in pressure and velocity.

Example 6.1.2. The following test shows convergence of the pressure projection step with pure non-homogeneous Dirichlet boundary conditions:

$$p = \cos(\pi(x^2 + y^2)) \qquad \text{(see Figure 6.1b)}$$
$$\vec{u}_{out} = \left(\frac{1}{2}x + \frac{1}{\sqrt{3}}y, \frac{1}{\sqrt{3}}x - \frac{1}{2}y\right)^\top, \qquad \text{(see Figure 6.2a)}$$

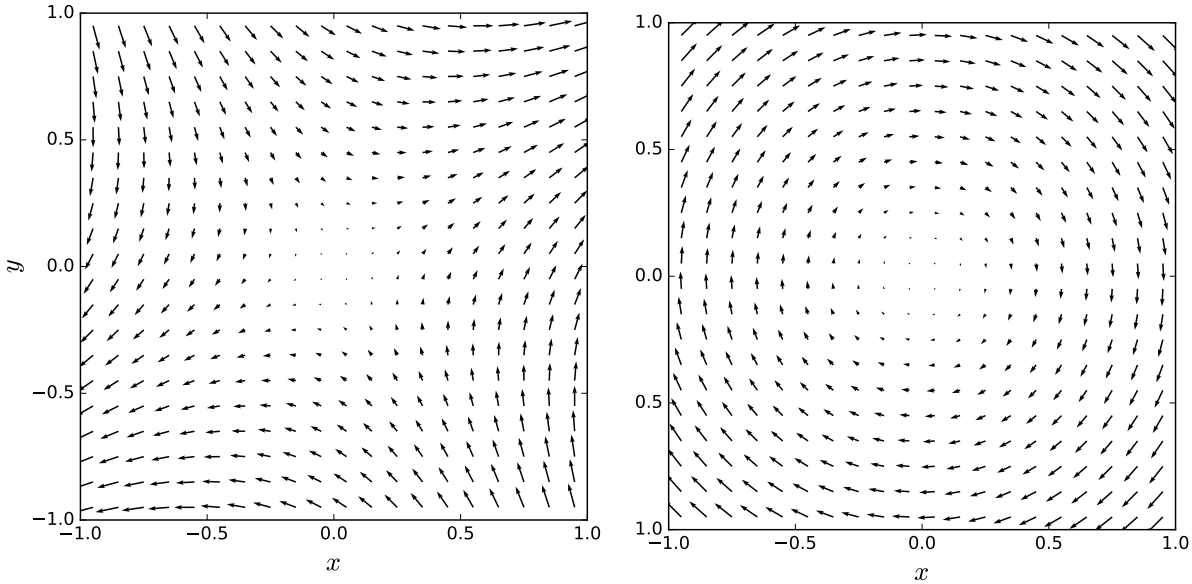
Table 6.2 demonstrates second order convergence in pressure and velocity.



(a) $p = 2((1+x)(1-x)(y+1)(1-y))^3 - 1$ **(b)** $p = \cos(\pi(x^2+y^2))$ **(c)** $p = \frac{1}{2}\left(\sin\left(\frac{\pi}{2}x\right) + \sin\left(\frac{\pi}{2}y\right)\right)$

(d) $p = (x^2+y^2)^2 \cos(3\,\mathrm{atan2}(y,x))$ **(e)** $f = 7(x^2+y^2)\cos(3\,\mathrm{atan2}(y,x))$

Figure 6.1: Four different pressure functions used to test the convergence properties of the MFD method applied to Poisson's problem, and one source function (bottom right). Functions on the bottom row are taken from the work of Johansen and Colella [1998] testing convergence of their Poisson solver.



(a) $\vec{u}_{out} = \left(\frac{1}{2}x + \frac{1}{\sqrt{3}}y, \frac{1}{\sqrt{3}}x - \frac{1}{2}y\right)^\top$ **(b)** $\vec{u}_{out} = (y, -x)^\top$

Figure 6.2: Two different output velocity functions used to test the convergence properties of the MFD method applied to Poisson's problem.

Grid	$\|\mathbf{p} - \mathbf{p}_h\|_\infty$	Order	$\|\mathbf{p} - \mathbf{p}_h\|_2$	Order	$\|\mathbf{p} - \mathbf{p}_h\|_1$	Order
32^2	2.66×10^{-3}		9.01×10^{-4}		7.24×10^{-4}	
64^2	6.63×10^{-4}	2.00	2.25×10^{-4}	2.00	1.82×10^{-4}	1.99
128^2	1.66×10^{-4}	2.00	5.63×10^{-5}	2.00	4.55×10^{-5}	2.00
256^2	4.14×10^{-5}	2.00	1.41×10^{-5}	2.00	1.14×10^{-5}	2.00
512^2	1.03×10^{-5}	2.00	3.52×10^{-6}	2.00	2.84×10^{-6}	2.00
1024^2	2.59×10^{-6}	2.00	8.79×10^{-7}	2.00	7.11×10^{-7}	2.00
2048^2	6.46×10^{-7}	2.00	2.20×10^{-7}	2.00	1.78×10^{-7}	2.00
4096^2	1.62×10^{-7}	2.00	5.50×10^{-8}	2.00	4.45×10^{-8}	2.00

Grid	$\|\mathbf{u}_{out} - \mathbf{u}_h\|_\infty$	Order	$\|\mathbf{u}_{out} - \mathbf{u}_h\|_2$	Order	$\|\mathbf{u}_{out} - \mathbf{u}_h\|_1$	Order
32^2	2.20×10^{-3}		1.24×10^{-3}		1.37×10^{-3}	
64^2	5.55×10^{-4}	1.98	3.11×10^{-4}	2.00	3.44×10^{-4}	1.99
128^2	1.39×10^{-4}	2.00	7.78×10^{-5}	2.00	8.62×10^{-5}	2.00
256^2	3.48×10^{-5}	2.00	1.94×10^{-5}	2.00	2.16×10^{-5}	2.00
512^2	8.70×10^{-6}	2.00	4.86×10^{-6}	2.00	5.39×10^{-6}	2.00
1024^2	2.18×10^{-6}	2.00	1.22×10^{-6}	2.00	1.35×10^{-6}	2.00
2048^2	5.44×10^{-7}	2.00	3.04×10^{-7}	2.00	3.37×10^{-7}	2.00
4096^2	1.36×10^{-7}	2.00	7.60×10^{-8}	2.00	8.43×10^{-8}	2.00

Table 6.1: Convergence of pressure $\mathbf{p}_h$ and velocity $\mathbf{u}_h$ in pressure projection with homogeneous Dirichlet boundary conditions for Example 6.1.1.

Grid	$\|\mathbf{p}-\mathbf{p}_h\|_\infty$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_2$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_1$	Order
32^2	6.98×10^{-3}		4.75×10^{-3}		4.09×10^{-3}	
64^2	1.74×10^{-3}	2.01	1.18×10^{-3}	2.01	1.02×10^{-3}	2.00
128^2	4.34×10^{-4}	2.00	2.95×10^{-4}	2.00	2.55×10^{-4}	2.00
256^2	1.09×10^{-4}	2.00	7.38×10^{-5}	2.00	6.37×10^{-5}	2.00
512^2	2.71×10^{-5}	2.00	1.84×10^{-5}	2.00	1.59×10^{-5}	2.00
1024^2	6.78×10^{-6}	2.00	4.61×10^{-6}	2.00	3.98×10^{-6}	2.00
2048^2	1.70×10^{-6}	2.00	1.15×10^{-6}	2.00	9.96×10^{-7}	2.00
4096^2	4.24×10^{-7}	2.00	2.88×10^{-7}	2.00	2.49×10^{-7}	2.00
Grid	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_\infty$	Order	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_2$	Order	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_1$	Order
32^2	3.07×10^{-2}		8.40×10^{-3}		7.09×10^{-3}	
64^2	7.73×10^{-3}	1.99	1.98×10^{-3}	2.08	1.68×10^{-3}	2.07
128^2	1.93×10^{-3}	2.00	4.81×10^{-4}	2.04	4.11×10^{-4}	2.04
256^2	4.83×10^{-4}	2.00	1.18×10^{-4}	2.02	1.01×10^{-4}	2.02
512^2	1.21×10^{-4}	2.00	2.94×10^{-5}	2.01	2.52×10^{-5}	2.01
1024^2	3.02×10^{-5}	2.00	7.32×10^{-6}	2.01	6.28×10^{-6}	2.00
2048^2	7.55×10^{-6}	2.00	1.83×10^{-6}	2.00	1.57×10^{-6}	2.00
4096^2	1.89×10^{-6}	2.00	4.56×10^{-7}	2.00	3.92×10^{-7}	2.00

Table 6.2: Convergence of pressure and velocity in pressure projection with non-homogeneous Dirichlet boundary conditions for Example 6.1.2.

6.1.2 Neumann Boundary Conditions

The following two examples deal with Neumann boundary conditions applied to the Poisson problem in a more general form.

Example 6.1.3. First we would like to test our discrete operators with Poisson's equation with homogeneous Neumann boundary conditions:

$$\begin{cases} \nabla^2 p = f & \text{in } \Omega \\ \nabla p \cdot \hat{n} = 0 & \text{on } \partial\Omega. \end{cases}$$

In particular, we will use the source term f to find the solution p:

$$f = -\frac{\pi^2}{8}\left(\sin\left(\tfrac{\pi}{2}x\right) + \sin\left(\tfrac{\pi}{2}y\right)\right)$$
$$p = \frac{1}{2}\left(\sin\left(\tfrac{\pi}{2}x\right) + \sin\left(\tfrac{\pi}{2}y\right)\right). \qquad \text{(see Figure 6.1c)}$$

Since we do not have a divergence operator on the right hand side, the MFD discretization for this problem is slightly different:

$$\mathbf{DGp} = \mathbf{f}.$$

It now depends on the matrix $\mathbf{M}_V$ (see Appendix A). We can observe second order convergence in pressure in Table 6.3.

Grid	$\|\mathbf{p}-\mathbf{p}_h\|_\infty$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_2$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_1$	Order
32^2	8.04×10^{-4}		4.21×10^{-4}		3.51×10^{-4}	
64^2	2.01×10^{-4}	2.00	1.03×10^{-4}	2.03	8.45×10^{-5}	2.05
128^2	5.02×10^{-5}	2.00	2.54×10^{-5}	2.02	2.07×10^{-5}	2.03
256^2	1.25×10^{-5}	2.00	6.31×10^{-6}	2.01	5.14×10^{-6}	2.01
512^2	3.14×10^{-6}	2.00	1.57×10^{-6}	2.00	1.28×10^{-6}	2.01
1024^2	7.84×10^{-7}	2.00	3.93×10^{-7}	2.00	3.19×10^{-7}	2.00
2048^2	1.96×10^{-7}	2.00	9.81×10^{-8}	2.00	7.96×10^{-8}	2.00
4096^2	4.93×10^{-8}	1.99	2.46×10^{-8}	1.99	2.00×10^{-8}	1.99

Table 6.3: Convergence of pressure in Poisson's equation with *homogeneous* Neumann boundary conditions for Example 6.1.3.

Grid	$\|\mathbf{p}-\mathbf{p}_h\|_\infty$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_2$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_1$	Order
32^2	1.11×10^{-2}		4.92×10^{-3}		3.94×10^{-3}	
64^2	2.78×10^{-3}	2.00	1.19×10^{-3}	2.05	9.41×10^{-4}	2.06
128^2	6.96×10^{-4}	2.00	2.93×10^{-4}	2.02	2.30×10^{-4}	2.03
256^2	1.74×10^{-4}	2.00	7.26×10^{-5}	2.01	5.69×10^{-5}	2.02
512^2	4.35×10^{-5}	2.00	1.81×10^{-5}	2.01	1.41×10^{-5}	2.01
1024^2	1.09×10^{-5}	2.00	4.51×10^{-6}	2.00	3.52×10^{-6}	2.00
2048^2	2.72×10^{-6}	2.00	1.13×10^{-6}	2.00	8.80×10^{-7}	2.00
4096^2	6.80×10^{-7}	2.00	2.81×10^{-7}	2.00	2.20×10^{-7}	2.00

Table 6.4: Convergence of pressure in Poisson's equation with *non-homogeneous* Neumann boundary conditions for Example 6.1.4.

Example 6.1.4. Now we want to solve Poisson's problem with non-homogeneous Neumann boundary conditions:

$$\begin{cases} \nabla^2 p = f & \text{in } \Omega \\ \nabla p \cdot \hat{n} = g^N & \text{on } \partial\Omega. \end{cases}$$

In particular, we will use the source term $f = 7r^2\cos(3\theta)$ (see Figure 6.1e) to get the solution $p = r^4\cos(3\theta)$ (see Figure 6.1d) with

$$\nabla p = r^2(4x\cos(3\theta) + 3y\sin(3\theta), -3x\sin(3\theta) + 4y\cos(3\theta))^\top,$$

where $r^2 = x^2 + y^2$ and $\theta = \operatorname{atan2}(y, x)$. The Neumann boundary condition is defined by $g^N = \hat{n} \cdot \nabla p$ on the boundary. Note that g^N is not defined at the corners of our domain since the normals there are undefined. Numerically, however, we can enforce vertical and horizontal boundary conditions simultaneously at the corners. This is made precise in Appendix A. We can once again observe second order convergence in Table 6.4.

Example 6.1.5. Using the configuration from Example 6.1.4 with

$$\vec{u}_{out} = \left(\frac{1}{2}x + \frac{1}{\sqrt{3}}y, \frac{1}{\sqrt{3}}x - \frac{1}{2}y\right)^\top, \qquad \text{(see Figure 6.2a)}$$

Grid	$\|\mathbf{p}-\mathbf{p}_h\|_\infty$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_2$	Order	$\|\mathbf{p}-\mathbf{p}_h\|_1$	Order
32^2	2.36×10^{-3}		8.63×10^{-4}		6.14×10^{-4}	
64^2	5.90×10^{-4}	2.00	2.08×10^{-4}	2.06	1.46×10^{-4}	2.08
128^2	1.47×10^{-4}	2.00	5.09×10^{-5}	2.03	3.54×10^{-5}	2.04
256^2	3.69×10^{-5}	2.00	1.26×10^{-5}	2.01	8.74×10^{-6}	2.02
512^2	9.22×10^{-6}	2.00	3.13×10^{-6}	2.01	2.17×10^{-6}	2.01
1024^2	2.30×10^{-6}	2.00	7.81×10^{-7}	2.00	5.40×10^{-7}	2.00
2048^2	5.76×10^{-7}	2.00	1.95×10^{-7}	2.00	1.35×10^{-7}	2.00
4096^2	1.44×10^{-7}	2.00	4.88×10^{-8}	2.00	3.37×10^{-8}	2.00
Grid	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_\infty$	Order	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_2$	Order	$\|\mathbf{u}_{out}-\mathbf{u}_h\|_1$	Order
32^2	9.20×10^{-4}		4.10×10^{-4}		4.46×10^{-4}	
64^2	2.31×10^{-4}	2.00	9.95×10^{-5}	2.04	1.07×10^{-4}	2.05
128^2	5.77×10^{-5}	2.00	2.45×10^{-5}	2.02	2.63×10^{-5}	2.03
256^2	1.44×10^{-5}	2.00	6.07×10^{-6}	2.01	6.53×10^{-6}	2.01
512^2	3.61×10^{-6}	2.00	1.51×10^{-6}	2.01	1.62×10^{-6}	2.01
1024^2	9.02×10^{-7}	2.00	3.77×10^{-7}	2.00	4.05×10^{-7}	2.00
2048^2	2.25×10^{-7}	2.00	9.43×10^{-8}	2.00	1.01×10^{-7}	2.00
4096^2	5.61×10^{-8}	2.01	2.36×10^{-8}	2.00	2.53×10^{-8}	2.00

Table 6.5: Convergence of pressure and velocity in pressure projection with non-homogeneous Neumann boundary conditions for Example 6.1.5.

we proceed as previously with non-homogeneous Neumann boundary conditions. Table 6.5 shows second order convergence in pressure and velocity.

6.2 Numerical Results on Irregular Domains

We have shown convergence of pressure and velocity provided by the MFD method on regular domains. This is not sufficient to simulate liquids with irregular deforming free surfaces, and gives nothing more than an alternative formulation to the MAC approach on regular grids. The advantage of using the MFD method is the ability to apply it on irregular domains with polygonal (or polyhedral in 3D) cells. As noted before, there are theoretical restrictions on the shapes of these cells, but they are notably weaker than restrictions imposed on triangulation quality required by finite-element methods or orthogonality constraints on finite-volume meshes.

From this section onward, we use the full C++ simulator for our tests. This section merely verifies the validity of our MFD pressure projection step on irregular domains, thus surface tension and surface tracking are omitted here. To summarize, we use a simple cut-cell mesh as our discrete domain. The interior of the domain is represented by regular square cells as shown in Figure 3.3a. The cells on the surface of the domain are computed using a marching-squares-like algorithm as described in Chapter 5 in detail; see Figure 5.1

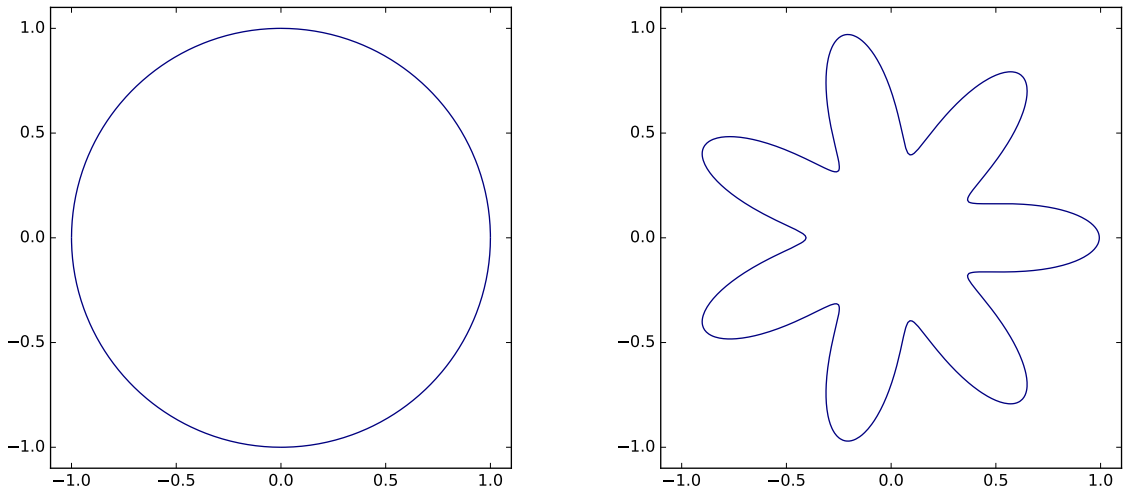
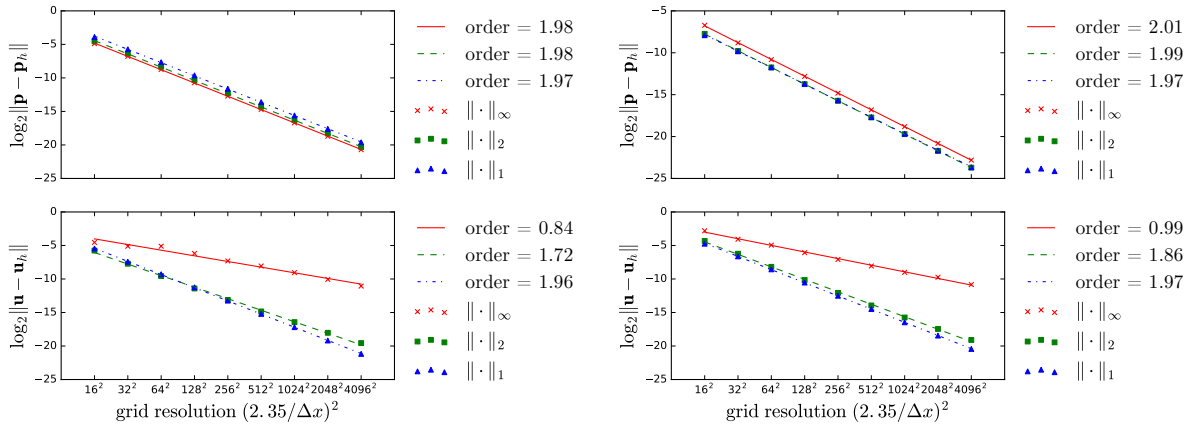


Figure 6.3: Irregular domains used to test convergence of pressure projection.

for a quick review. Of course, to be consistent with the mesh that we use for fluid simulation, we also collapse small interior edges as described in Section 5.3. We emphasize that this step is not required to see numerical convergence; it is primarily used to eliminate degenerate cells generated by a moving surface mesh.

We use two irregular domains to test the convergence of pressure projection using the MFD approach. The first is a unit circle centered at the origin as in Figure 6.3a. The second, is a seven arm flower depicted in Figure 6.3b, which, in general, is given by the zero level set of the function $\phi = x^2 + y^2 - r^2(1 - a\sin(n\,\mathrm{atan2}(x, y)))^2$, where r is the radius, $a \in [0, 1)$ is the amplitude and $n \in \mathbb{N}$ is the number of arms. For reference, the zero level set of ϕ can be computed by the parametrized curve $(x(\theta), y(\theta))$ where $x(\theta) = r(1 - a\sin(n\theta))\sin(\theta)$ and $y(\theta) = r(1 - a\sin(n\theta))\cos(\theta)$. For the following examples, we take $n = 7$, $a = 0.42$ and $r = 0.7$, where it fits neatly in the unit circle. The flower geometry is often used for convergence tests on irregular boundaries [Gibou et al., 2002; Popinet, 2003; Johansen and Colella, 1998]. The following examples test the convergence rate of the pressure and velocity of the MFD method applied on an irregular domain. We expect second order convergence in pressure and first order convergence in velocity [Lipnikov et al., 2014].

Example 6.2.1 (Dirichlet Boundary Conditions)**.** We use the test functions from Example 6.1.2 to verify convergence of pressure projection, but now on two irregular domains as



(a) **(b)**

Figure 6.4: Circle (a) and flower (b) convergence test with non-homogeneous Dirichlet boundary conditions for Example 6.2.1. Using the log-log plots to compare error in velocity $\|\mathbf{u}-\mathbf{u}_h\|$ and pressure $\|\mathbf{p}-\mathbf{p}_h\|$ against the grid resolution. We find the order of convergence from the slope of the line of mean least squares fit. For the circle domain (a) for instance, the order of convergence is 1.98 for pressure and 0.84 for velocity in the ∞-norm.

previously mentioned. Figure 6.4 shows expected convergence results for Dirichlet boundary conditions.

Example 6.2.2 (Neumann Boundary Conditions)**.** We use the configuration from Example 6.1.4 testing convergence with non-homogeneous Neumann boundary conditions. Figure 6.5 shows second order convergence in pressure and first order convergence in velocity as expected.

Mixed boundary conditions are the most general boundary conditions we need to handle to simulate fluids. The following example verifies the convergence of our Poisson solver subject to Neumann boundary conditions on $\Gamma^N \subset \partial\Omega$ and Dirichlet boundary conditions on $\Gamma^D \subset \partial\Omega$.

Example 6.2.3 (Mixed Boundary Conditions)**.** Here we reuse the previous test functions as in Example 6.1.4. The boundary is split into $\Gamma^N = \{\,(x,y)^\top \in \partial\Omega \;:\; y < 0\,\}$ and $\Gamma^D = \{\,(x,y)^\top \in \partial\Omega \;:\; y \geq 0\,\}$ akin to a liquid exposed to air on the top with a solid boundary on the bottom. Figure 6.6 shows expected convergence results for mixed boundary conditions.

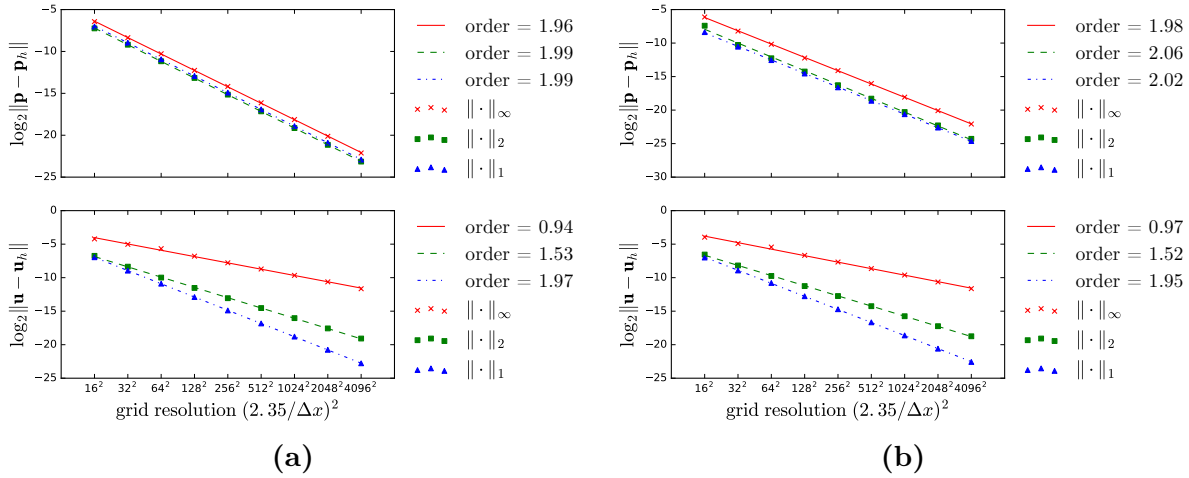


Figure 6.5: Circle (a) and flower (b) convergence test with pure non-homogeneous Neumann boundary conditions for Example 6.2.2. As in Figure 6.4 we use the log-log plots to compare error in pressure and velocity. For the circle domain (a) for instance, the order of convergence is 1.96 for pressure and 0.94 for velocity in the ∞-norm.

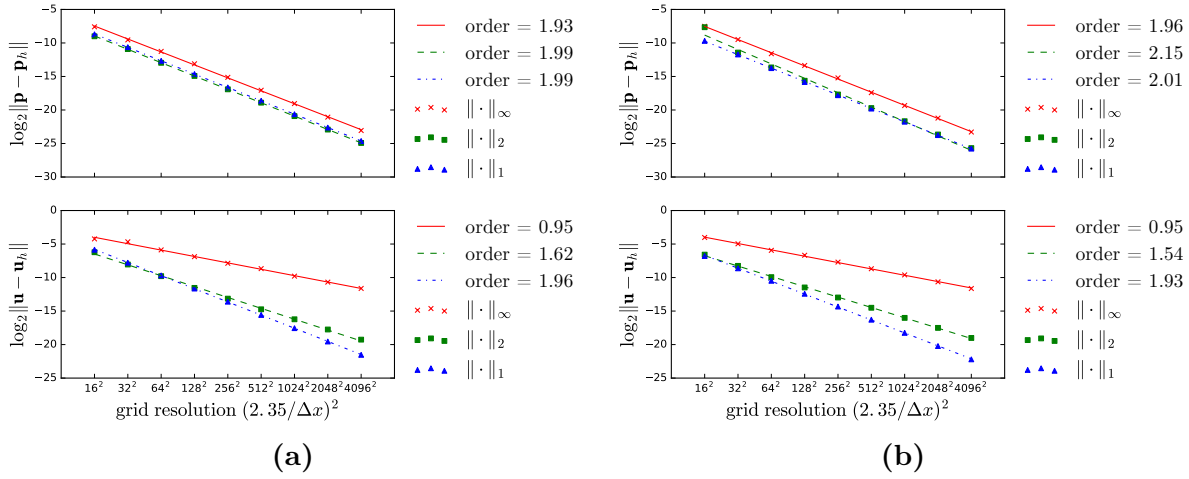


Figure 6.6: Circle (a) and flower (b) convergence tests with mixed boundary conditions for Example 6.2.3. As in Figure 6.4 we use the log-log plots to compare error in pressure and velocity. For the circle domain (a) for instance, the order of convergence is 1.93 for pressure and 0.95 for velocity in the ∞-norm.

6.3 Example Liquid Simulations

This section demonstrates all elements of our fluid simulator. We illustrate pressures within the liquid using a relative blue-green-red colour scale representing low-medium-high pressures respectively. We begin with the evolution of two droplets in an off-center collision as shown in Figure 6.9. This demonstration shows that our method is already able to simulate droplets pinching off due to Plateau-Rayleigh instability without an explicit treatment of mesh merging and splitting [Quan et al., 2009]. Topology changes here are handled automatically since we reconstruct the surface mesh at every time step using the marching squares templates as described in Chapter 5.

To assess the accuracy of our surface tension model, consider the oscillating droplet test [Enright et al., 2003]. A droplet with interface defined in polar coordinates by $r = a(1 + s\cos(n\theta))$ is subject to surface tension forces. The frequency of oscillation of said droplet, ω, is given by $\omega^2 = 6\gamma/a^3$. To set the period to π, we let the surface tension coefficient $\gamma = 2/81$ and $a = 1/3$. Our method correctly maintains an accurate period of oscillation $T = 2\pi/\omega$ within one or two time steps. For instance, given $\Delta t = 0.04$, the mean oscillation for the first 25 oscillations is 3.18 for grids of size 64^2, 128^2 and 256^2. After a certain number of oscillations (e.g. 25 in this test) numerical dissipation effects become significant. We can further demonstrate that the amplitude of oscillation approaches the true amplitude of 0.35 as we increase the resolution as shown in Figure 6.7.

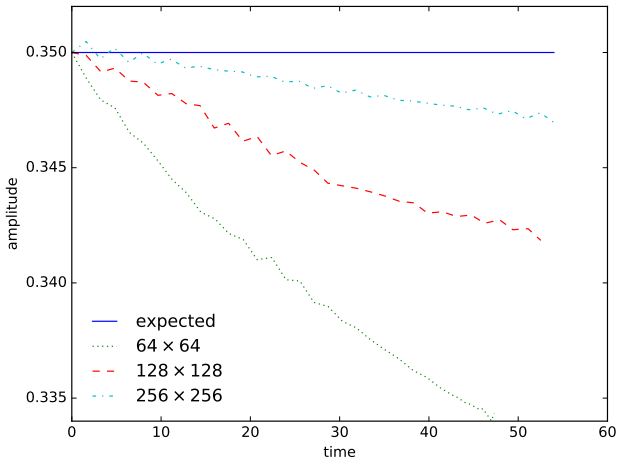


Figure 6.7: Amplitude of an oscillating droplet approaches the expected amplitude as we increase the grid resolution.

To show that our method maintains symmetry even in configurations not aligned with the grid, consider Figure 6.8. Here we demonstrate a lack of grid artifacts and symmetry preservation on a simulation on a fairly coarse grid and a relatively large time step. Note that this example (Figure 6.8b) also demonstrates dampening of liquid motion, which can be attributed to semi-Lagrangian velocity advection.

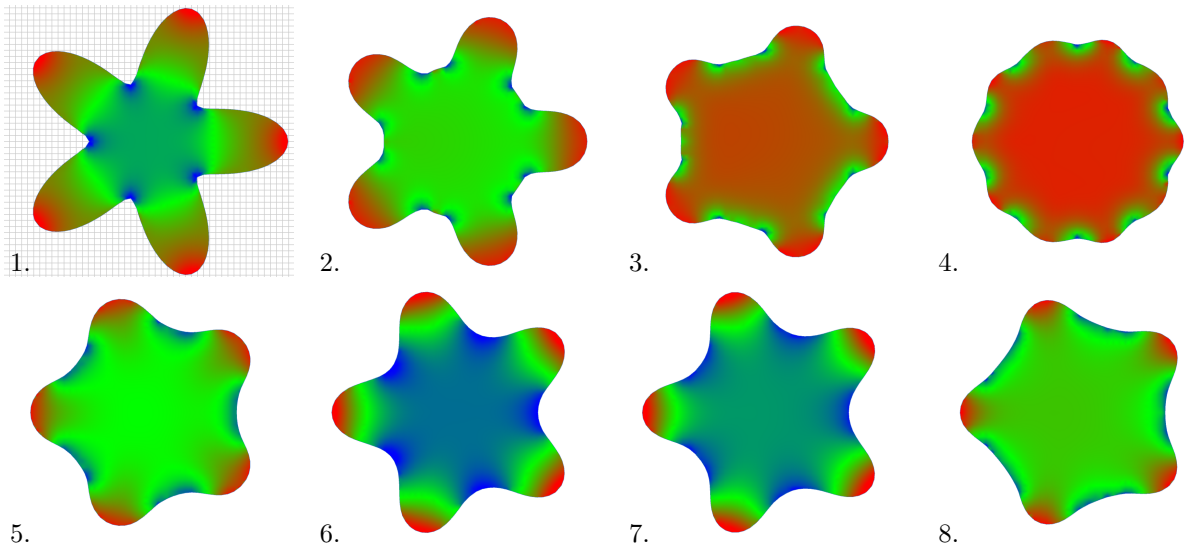
Next, we demonstrate our treatment of contact angles by dropping a liquid droplet onto a flat surface in Figures 6.10 and 6.11 (to reduce visual noise, we use a lighter blue color to indicate areas with lower pressure). More specifically, Figure 6.10 shows a liquid droplet interacting with a hydrophilic surface. Figure 6.10a shows a surface so hydrophilic that the liquid covers the entire bottom of the container. Figure 6.10b shows a droplet[1] spread across a portion of the container making a 70° angle with the solid surface. Figure 6.11b shows a hydrophobic surface, that strongly repels liquid, causing it to bounce once. Finally, Figure 6.11a demonstrates a surface that is neither hydrophilic nor hydrophobic, making a 90° angle with the liquid surface.

Lastly, we show the use of irregular boundaries along with one way liquid-solid coupling using a kinematic solid[2] moving through the liquid as depicted in Figure 6.12. To achieve this we set the velocity of the liquid at the solid boundary to be the same as the velocity of the solid normal to the boundary. This test demonstrates proper use of non-homogeneous Neumann boundary conditions as the liquid flow follows the kinematic solid. The lack of splashes can be attributed to numerical dissipation in semi-Lagrangian advection. Introducing surface tension and making solids hydrophobic as in Figure 6.13, produces interesting results, which encompass all components of our liquid simulator. Each time step of the simulation in Figure 6.13 took a maximum of 0.35 s and an average of 0.29 s. The pressure projection step in the same test took a maximum of 0.28 s and an average of 0.23 s per time step, thus it constitutes the majority of computation time as expected.

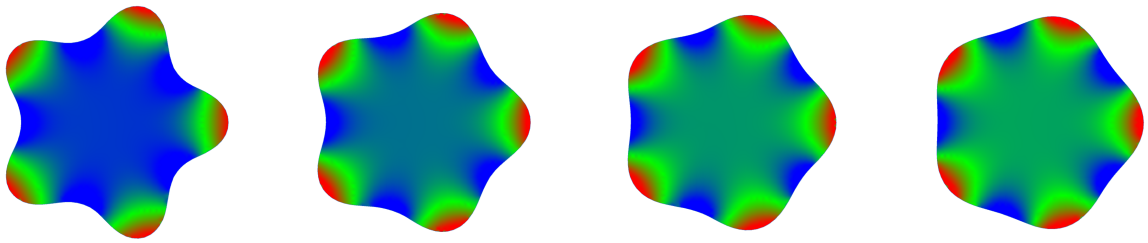
Since our implementation is restricted to 2D flows, unfortunately the simulations can't be compared to realistic water-like liquids. However, our results demonstrate the flexibility of the simulator and all the features discussed in this thesis. With a 3D implementation coupled with a less dissipative velocity advection scheme and a careful choice of parameters, one can start to compare simulated effects with real liquids.

[1] Out of the four tests, this is the only test in which the droplet did not have some initial downward velocity to prevent it from touching the sides of the container.

[2] A kinematic solid is an object with infinite mass that moves along a fixed trajectory.



(a) The simulation of the first oscillation shown above maintains symmetry on a fairly coarse grid with large time steps.



(b) Oscillatory motion dampens after the first 4 full oscillations shown above.

Figure 6.8: Oscillating five-arm flower simulated on a 64×64 grid with $\Delta x = 0.04$, surface tension coefficient $\gamma = 0.005$, and time step $\Delta t = 0.04\,\mathrm{s}$.

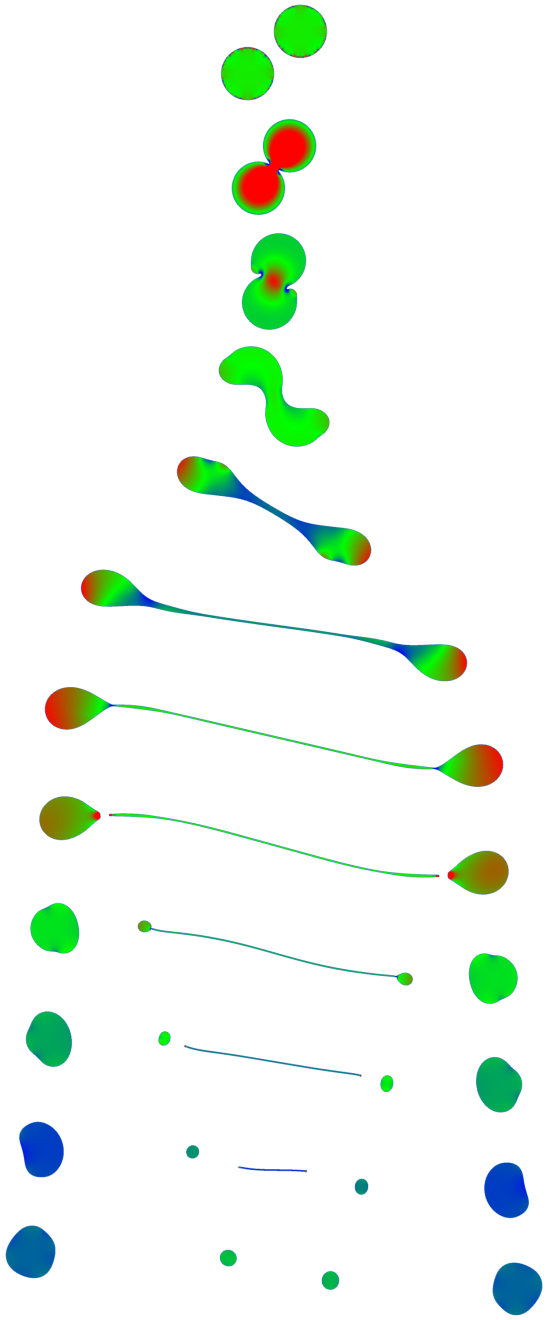
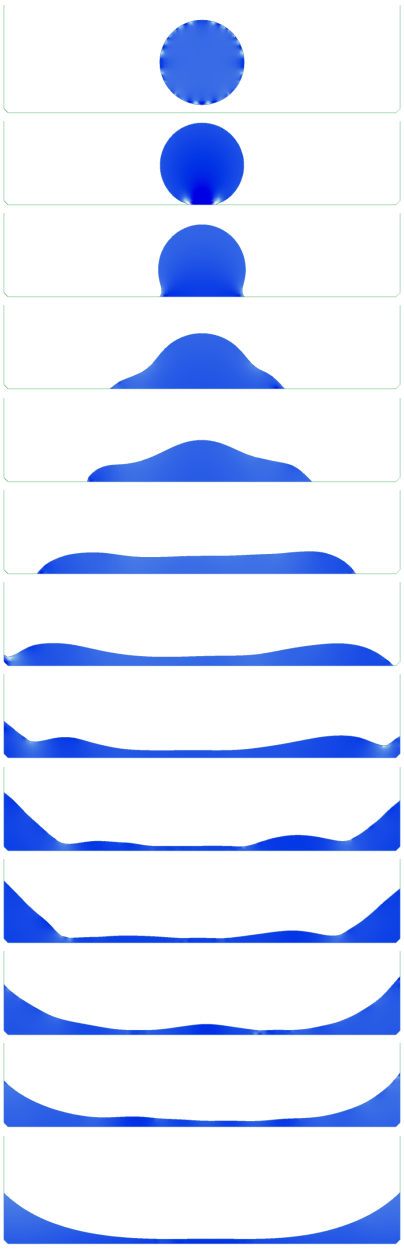
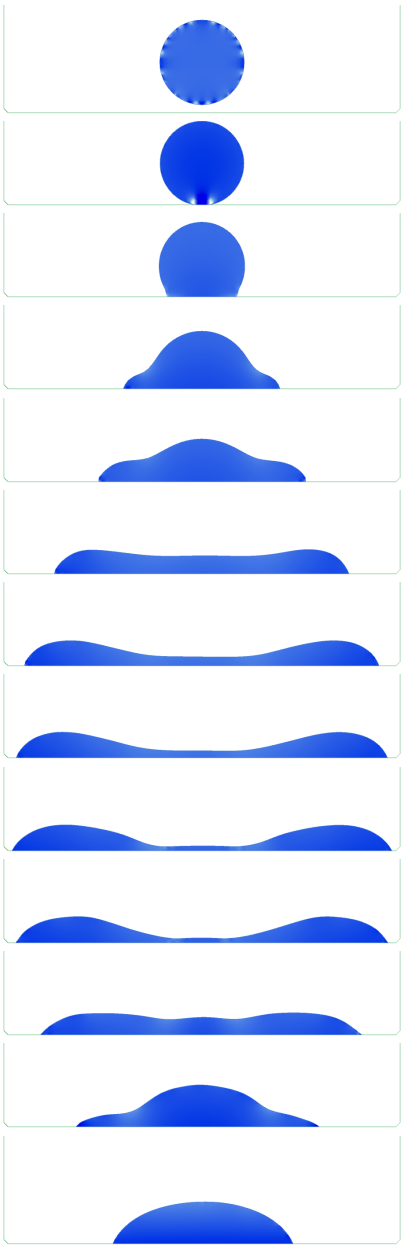


Figure 6.9: Off-center collision between two droplets. For this simulation we used a 512×512 grid with $\Delta x = 0.005$, surface tension coefficient $\gamma = 0.0018$, and time step $\Delta t = 0.007\,\text{s}$. The initial relative velocity of the droplets is $u_0 = 0.9$ m/s.

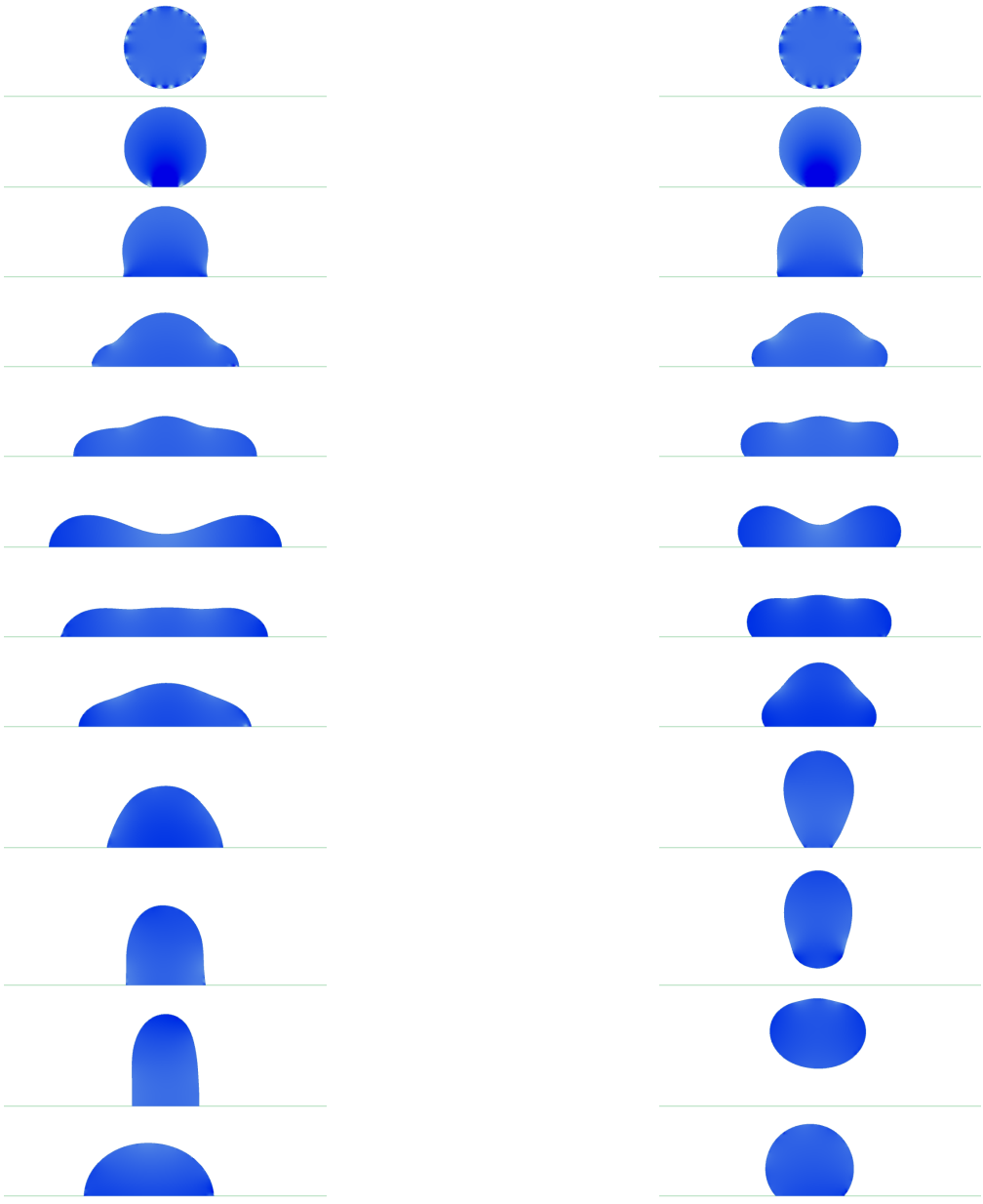


(a) The surface is very hydrophilic making a contact angle $\theta_s = 45°$ with the liquid surface.



(b) The solid surface makes a contact angle $\theta_s = 70°$ with the liquid surface.

Figure 6.10: Droplet falling on two different hydrophilic solid surfaces. For this simulation we used a 128×128 grid with $\Delta x = 0.02$, surface tension coefficient $\gamma = 0.02$, and time step $\Delta t = 0.01\,\text{s}$. Both figures show snapshots at times $t = 0$, 0.1, 0.2, 0.4, 0.6, 1.1, 1.7, 2.0, 2.7, 3.0, 4.0, 4.8 and 15.0 seconds.



(a) The surface is neither hydrophobic nor hydrophilic, i.e. the contact angle between the liquid and solid surface is $\theta_s = 90°$. Frames at times $t = 0.0$, 0.1, 0.2, 0.4, 0.6, 1.3, 2.0, 2.3, 2.5, 2.8, 3.0, and 20.2 seconds are shown.

(b) The solid surface is hydrophobic making an angle $\theta_s = 140°$ with the liquid. Frames at times $t = 0.0$, 0.1, 0.2, 0.4, 0.6, 1.0, 1.3, 1.5 2.0, 2.5, 3.0, and 20.2 seconds are shown.

Figure 6.11: Droplet falling on two different solid surfaces. Here we used the same configuration as in Figure 6.10. Different frames are shown in each of the above examples to outline features specific to each case.

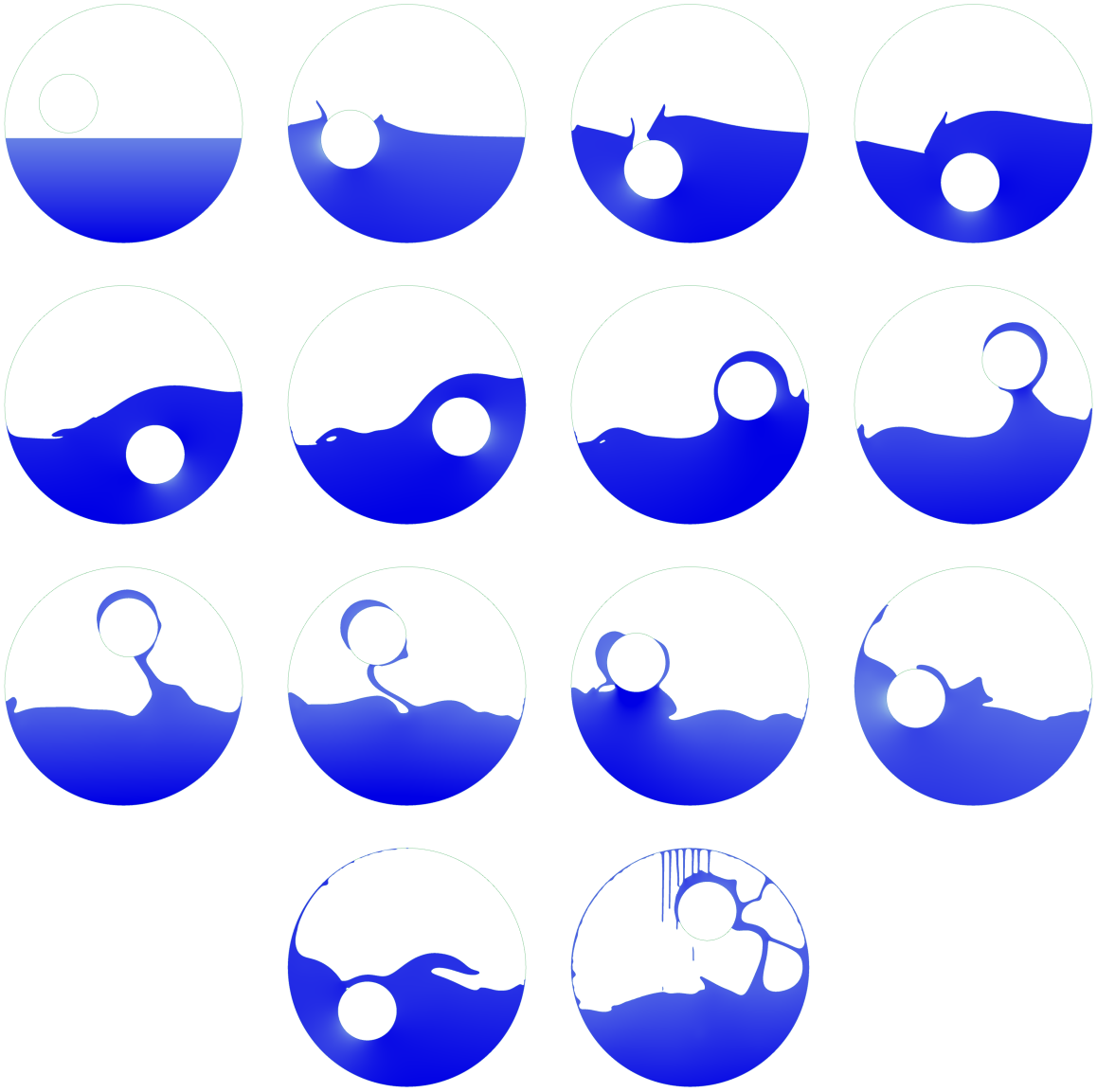


Figure 6.12: A rotating kinematic solid circle is moved through the liquid resting inside a circular container. This simulation was performed on 256×256 grid with $\Delta x = 0.01$, and time step $\Delta t = 0.02\,\mathrm{s}$. Snapshots were taken on the second for 12 seconds as well as at $t = 189.5\,\mathrm{s}$ for the last snapshot.

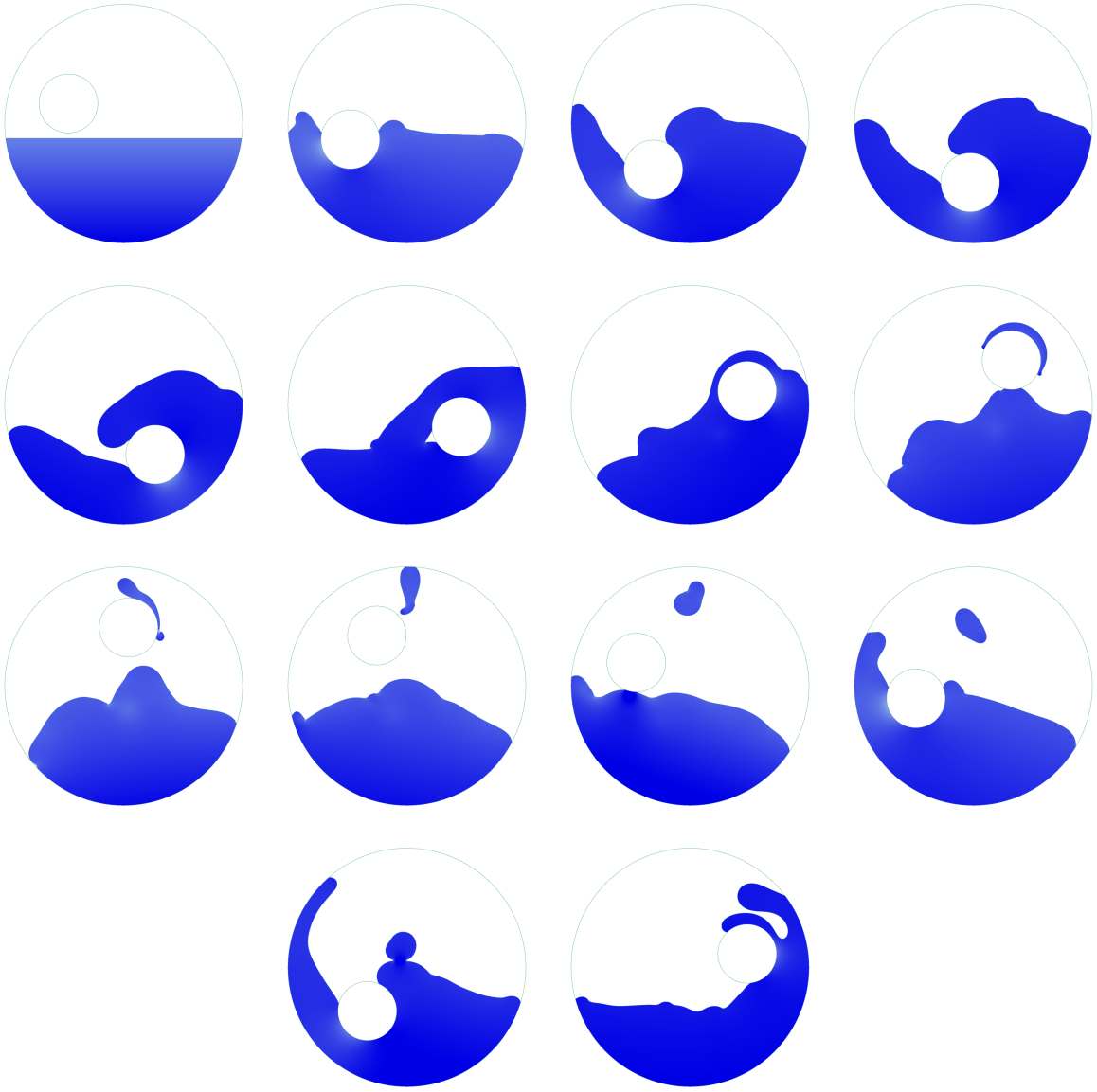


Figure 6.13: A rotating kinematic hydrophobic solid circle is moved through a liquid with high surface tension, resting in a hydrophobic circular container. This simulation was performed on 256×256 grid with $\Delta x = 0.01$, surface tension coefficient $\gamma = 0.001$, and time step $\Delta t = 0.02\,\mathrm{s}$. Snapshots were taken on the second for 12 seconds as well as at $t = 107.0\,\mathrm{s}$ for the last snapshot.

Chapter 7

Conclusion

Our goal was to build a unified method to simulate water-like liquids that allows for seamless coupling with an accurate surface tension formulation able to handle specific contact angles. After iterating over numerous prototypes, we settled on the method presented in this thesis. Our original approach for pressure projection involved the MFD method with a slightly different discretization. More specifically, velocity samples normal to cell faces were stored at face midpoints and pressures were stored at cell centroids as well as face midpoints. Figure 7.1 illustrates this discretization on the interior of the domain. We used a face offsetting technique for moving the surface through the grid [Jiao, 2007]. This allowed for explicit control over Dirichlet as well as Neumann boundary conditions since pressure and normal velocity samples were stored on the surface. In addition, this approach exhibited insensitivity to small angles and small edges in the mesh. However, we chose to abandon this approach for pressure projection for two reasons. Primarily, the pressure solve under this discretization was in hydrostatic equilibrium under gravity, even when the surface contained high frequency features as in Figure 7.2. This was due to only having pressure samples at face midpoints and not at surface vertices. Additionally, the simulation was generally over-damped. We conjecture that the lack of pressure and velocity degrees of freedom on surface vertices caused sharp features to persist on the surface geometry. Furthermore, the force of surface tension, while computed on

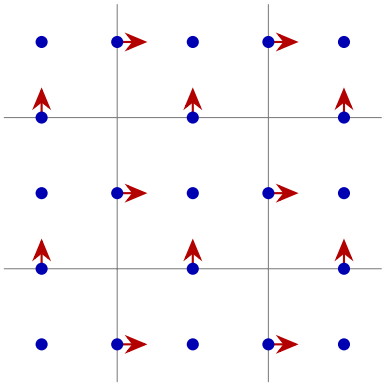


Figure 7.1: Original MFD discretization with pressure samples at cell centroids, as well as both pressure samples and velocity components normal to each face at face centres.

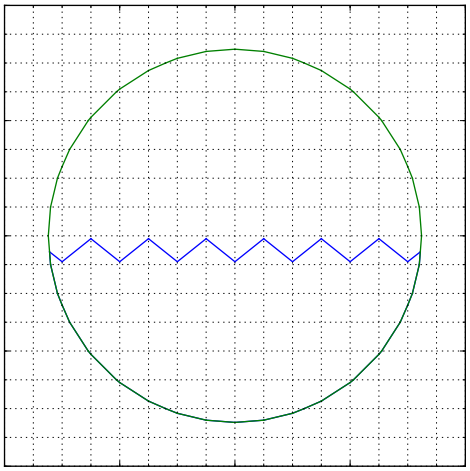


Figure 7.2: An example of sharp features between grid cells that our original MFD discretization failed to smooth in the presence of a gravitational force due to a lack of degrees of freedom on mesh vertices.

surface vertices, did not cause a velocity change on the vertices directly. Instead, the force was interpolated to the faces, causing a smoothing in the propagation of the surface.

7.1 Future Work

Since our work is ultimately a prototype for a 3D simulation, an immediate extension is a 3D implementation. Two key areas in which changes would be required to achieve a successful 3D extension are the formula for the mass matrix $\mathbf{M}_E$, which is given in [Veiga et al., 2014], and the handling of the contact curve with clipping. We have already mentioned that our mesh relaxation technique ought to extend to 3D meshes without much extra work, while the 3D extension of all other parts of our method follow trivially or are identical. It is possible to reduce the complexity of clipping within a computational cell if we reduce to tetrahedral cells near the surface. Note that we still gain flexibility using the MFD method here since we do not need to tetrahedralize clipped cells before the pressure projection step.

Independently of a 3D implementation, we would like to see this method coupled with a more ubiquitous advection scheme such as PIC/FLIP or PIC/APIC [Zhu and Bridson, 2005; Jiang et al., 2015]. This would produce more dynamic flows but would also require a method to extract an explicit surface mesh from a cloud of particles. An existing PIC/FLIP

fluid solver would typically already have a method for skinning a cloud of particles since it is often needed for rendering. Alternatively, we could use our current method for surface mesh evolution, and simply use particles for velocity advection.

Our framework can be extended to support other physical phenomena. For instance high or variable viscosity could be supported by extending the MFD method to full incompressible Navier-Stokes. In addition, it would be valuable to handle the no-penetration inequality type Neumann boundary condition with $\vec{u} \cdot \hat{n} \geq 0$, where $\hat{n}$ is the outward facing normal to the solid surface and $\vec{u}$ is the velocity field on the fluid. This boundary condition would allow liquid to peel off surfaces, which may be desirable for some scenarios of visual interest (e.g. last frame in Figure 6.12), although this conflicts with *true* fluid behaviour [Batty et al., 2007].

7.2 Limitations

As our approach shows potential for extensions, it also presents a few difficulties. For instance, one major flaw of the semi-implicit surface tension scheme is the artificial deformation it introduces to a circular liquid surface moving through the grid as shown on the top of Figure 5.2. The surface energy estimate for the predicted mesh configuration at the next time step is coupled with the incompressibility solve for the mesh configuration at the current time step. This mismatch can be seen in equation 4.7, and is the culprit of the artificial deformation in the moving droplet. Fully implicit schemes [Zheng et al., 2015] have been introduced to alleviate this issue, but are currently fairly costly to compute.

Other issues stem from the surface tracking nature in our approach. Primarily the liquid surface incident to a solid boundary can produce very shallow angles. These angles may cause the liquid to get "stuck" to the surface of the solid. Moreover, when coming to a rest, the simulation can be unstable at times when the resting point of the surface mesh lies close to the intersection of the solid surface and grid edge. We may be able to resolve such instabilities by introducing extra degrees of freedom at the contact vertices, or by using an adaptive higher order MFD discretization in those areas akin to the work of Edwards and Bridson [2014].

APPENDICES

Appendix A

MFD Discretization of the Poisson Problem

In this appendix, we will cover the missing details behind the mimetic discretization in Section 3.1. Consider once again Poisson's problem presented in (2.4). We mentioned in Section 3.1 that the MFD framework allows us to discretize this PDE directly using our discrete divergence and gradient operators in equation 3.7. Unfortunately, this framework does not describe a general way to discretize boundary conditions. Thus we need a closer look at the continuous PDE subject to boundary conditions, which is also called a boundary value problem (BVP). In our case, this is just the PDE (2.4) subject to boundary conditions (2.5), reiterated once more for reference:

$$\Delta t \nabla \cdot \nabla p = \nabla \cdot \vec{u}^{**} \qquad \text{in } \Omega \tag{A.1a}$$

$$p = f^D \qquad \text{on } \Gamma^D \tag{A.1b}$$

$$\hat{n} \cdot \Delta t \nabla p = \hat{n} \cdot \vec{u}^{**} - f^N \qquad \text{on } \Gamma^N. \tag{A.1c}$$

This BVP is a specialization of the more general form of the Poisson BVP:

$$\nabla \cdot \nabla p = f \qquad \text{in } \Omega \tag{A.2a}$$

$$p = g^D \qquad \text{on } \Gamma^D \tag{A.2b}$$

$$\hat{n} \cdot \nabla p = g^N \qquad \text{on } \Gamma^N, \tag{A.2c}$$

where $f = \Delta t^{-1} \nabla \cdot \vec{u}^{**}$, $g^D = f^D$, and $g^N = \Delta t^{-1}(\hat{n} \cdot \vec{u}^{**} - f^N)$ gives the original BVP (A.1). We will use the general Poisson BVP here to simplify our exposition.

Before we continue, it will be useful to classify a specific space of functions to make our exposition precise. First, consider the space of square integrable scalar functions on Ω defined by

$$L_2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \,\middle|\, \int_\Omega |f|^2 < \infty \right\}.$$

Then collect all such functions with square integrable partial derivatives:

$$H^1(\Omega) = \left\{ f \in L_2(\Omega) \,\middle|\, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \in L_2(\Omega) \right\}.$$

The space H^1 is a Sobolev space, and it characterizes solutions to the Poisson BVP. We will mainly be interested in functions from H^1 with a fixed value on the Dirichlet boundary Γ^D. That is for a particular function $g : \Gamma^D \to \mathbb{R}$, define

$$H^1_g = \left\{ f \in H^1(\Omega) \,\middle|\, f|_{\Gamma^D} = g \right\}.$$

These definitions will help us describe the duality of the continuous differential operators and write down the weak form of the Poisson BVP that we will discretize.

Indeed, any solution $p \in H^1_{g^D}$ to (A.2) also solves

$$\int_\Omega \nabla q \cdot \nabla p = \int_\Omega qf + \int_{\Gamma^N} qg^N \qquad \forall q \in H^1_0. \tag{A.3}$$

This is precisely the weak form of (A.2) as it imposes weaker constraints on the solution. Not unlike other methods for solving elliptic PDEs, we will focus on finding weak solutions by discretizing equation A.3. We will use discrete mimetic inner products to emulate continuous inner products.

A.1 Continuous Framework

Define two continuous inner products on the space of vector and scalar functions as follows [Hyman and Shashkov, 1998]. Let $\vec{v}, \vec{w} : \Omega \to \mathbb{R}^2$ be two vector functions, and $q, r : \Omega \to \mathbb{R}$ two scalar functions, then define inner products

$$(\vec{v}, \vec{w}) = \int_\Omega \vec{u} \cdot \vec{w} \tag{A.4}$$

$$(q, r) = \int_\Omega qr + \oint_{\partial\Omega} qr. \tag{A.5}$$

It will be clear which inner product is used by the dimensionality of the function codomain. If we extend the divergence operator to the boundary as

$$\mathcal{D}\,\vec{v} = \begin{cases} \nabla \cdot \vec{v} & \text{in } \Omega \\ -\vec{v} \cdot \hat{n} & \text{on } \partial\Omega \end{cases}. \tag{A.6}$$

Now to draw the duality between the continuous gradient and extended divergence operators, take a scalar function $q \in H^1$ and a vector function $\vec{v}$ as before, and consider the following integration by parts formula:

$$\int_\Omega \nabla q \cdot \vec{v} = \oint_{\partial\Omega} q(\vec{v} \cdot \hat{n}) - \int_\Omega q \nabla \cdot \vec{v}. \tag{A.7}$$

Using the inner product notation, we can rewrite this simply as

$$(\nabla q, \vec{v}) = -(q, \mathcal{D}\,\vec{v}) \tag{A.8}$$

giving us the expected duality relationship $\mathcal{D} = -\nabla^*$, which we aim to preserve in our discretization. If we also express the weak formulation (A.3) in terms of these inner products:

$$(\nabla q, \nabla p) = (q, F), \tag{A.9}$$

where naturally we define

$$F = \begin{cases} f & \text{in } \Omega \\ g^N & \text{on } \partial\Omega \end{cases}, \tag{A.10}$$

then by duality, $(\nabla q, \nabla p) = -(q, \nabla^* \nabla p)$, where $\nabla^* \nabla$ is symmetric and nonnegative [Hyman and Shashkov, 1998]. Preserving this property in the discrete Laplacian will allow us to use efficient iterative solvers like preconditioned conjugate gradient.

In the next section, we will construct the discrete scalar inner product to mimic (A.5). This will provide sufficient machinery to discretize the Poisson problem with Neumann boundary conditions.

A.2 Discrete Framework

We have covered Dirichlet boundary conditions on the discrete system in Chapter 3, so our goal now is to encode Neumann boundary conditions into the discrete framework. We will reuse the notation from Section 3.1 used to describe discrete quantities. First, we discretize the scalar inner product (A.5).

Naturally we will discretize scalar functions over the domain using values sampled at mesh nodes as pressure is in Section 3.1. Thus any two scalar functions $q, r : \Omega \to \mathbb{R}$ as before, are discretized with vectors $\mathbf{q}, \mathbf{r} \in \mathbb{R}^{N_V}$ where $\mathbf{q}_n = q(\vec{x}_n)$ and $\mathbf{r}_n = r(\vec{x}_n)$ at each node n on the mesh. In addition, to encode surface integrals into our inner product, scalar functions $g, h : \partial\Omega \to \mathbb{R}$ representing flow through the boundary, are discretized by vectors $\mathbf{g}, \mathbf{h} \in \mathbb{R}^{2N_{SV}}$, where $\mathbf{g}_{2i}$ and $\mathbf{g}_{2i+1}$ (resp. $\mathbf{h}_{2i}$ and $\mathbf{h}_{2i+1}$) represent the flow of g (resp. h) normal to edges $e_{i-1,i}$ and $e_{i,i+1}$ incident to vertex i respectively as depicted in Figure A.1. Splitting the flux at each vertex into two components like this will allow us to approximate an integral over the partial boundary, for instance $\Gamma^N \subset \partial\Omega$. This kind of flexibility is needed when discretizing the Poisson problem with mixed boundary conditions. Then we define vectors $\bar{\mathbf{q}}^\top = (\mathbf{q}^\top, \mathbf{g}^\top)$ and $\bar{\mathbf{r}}^\top = (\mathbf{r}^\top, \mathbf{h}^\top)$ containing interior and surface values. We will follow in a similar manner as (3.13) to define the discrete inner product as a sum of cell-based and edge-based inner products:

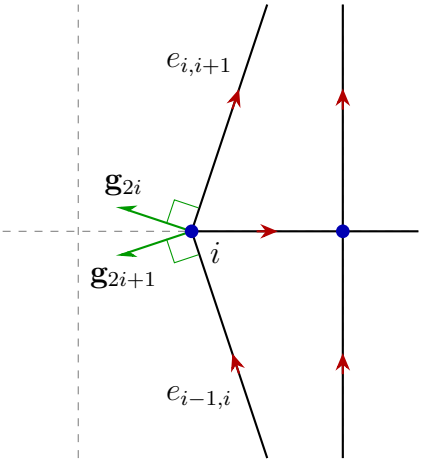


Figure A.1: Discretization of a function $g : \partial\Omega \to \mathbb{R}$, that has the form $g = \vec{v} \cdot \hat{n}$ for some vector field $\vec{v}$ defined on $\partial\Omega$ and $\hat{n}$ is the outward unit normal to the liquid surface. $\mathbf{g}_{2i}$ and $\mathbf{g}_{2i+1}$ encode the components of g in each normal direction.

$$\begin{aligned}\langle \bar{\mathbf{q}}, \bar{\mathbf{r}} \rangle_{\bar{V}} &= \langle \mathbf{q}, \mathbf{r} \rangle_V + \langle \mathbf{g}, \mathbf{h} \rangle_{SV} \\ &= \sum_c \langle \mathbf{q}_c, \mathbf{r}_c \rangle_{V,c} + \sum_{e \in \partial\Omega} \langle \mathbf{g}_e, \mathbf{h}_e \rangle_{SV,e} \\ &= \sum_c \mathbf{q}_c^\top \mathbf{M}_{V,c} \mathbf{r}_c + \sum_{e \in \partial\Omega} \mathbf{g}_e^\top \mathbf{M}_{SV,e} \mathbf{h}_e \quad \forall \bar{\mathbf{q}}, \bar{\mathbf{r}} \in \mathbb{R}^{N_V + N_{SV}},\end{aligned}$$

where $\mathbf{q}_c$ and $\mathbf{r}_c$ are the vectors $\mathbf{q}$ and $\mathbf{r}$ restricted to the nodes of cell c, while $\mathbf{g}_e$ and $\mathbf{h}_e$ are the vectors $\mathbf{g}$ and $\mathbf{h}$ restricted to the edge e. Local inner product matrices $\mathbf{M}_{SV,e}$ and $\mathbf{M}_{V,c}$ will define the global inner product, which can be written in terms of the global inner product matrix $\bar{\mathbf{M}}_V$ as

$$\langle \bar{\mathbf{q}}, \bar{\mathbf{r}} \rangle_{\bar{V}} = \bar{\mathbf{q}}^\top \begin{pmatrix} \mathbf{M}_V & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{SV} \end{pmatrix} \bar{\mathbf{r}} = \bar{\mathbf{q}}^\top \bar{\mathbf{M}}_V \bar{\mathbf{r}}, \tag{A.11}$$

where $\mathbf{M}_V$ is precisely the inner product matrix from (3.4). Our goal is to approximate the integrals from the continuous inner product using this discrete inner product as $\langle \bar{\mathbf{q}}, \bar{\mathbf{r}} \rangle_{\bar{V}} \approx (q, r)$. Starting with the local cell integral we approximate

$$\langle \mathbf{q}_c, \mathbf{r}_c \rangle_{V,c} \approx \int_c qr. \tag{A.12}$$

We use a simple quadrature rule to discretize this integral as

$$\int_c qr \approx \sum_{n \in \partial c} w_{c,n} \mathbf{q}_n \mathbf{r}_n,$$

where $n \in \partial c$ collects all nodes n on the boundary of cell c. The weights $w_{c,n}$ can be taken from the centroid formula

$$\vec{x}_c = \sum_{n \in \partial c} w_{c,n} \vec{x}_n.$$

The resulting matrix $\mathbf{M}_V$ is diagonal and if the weights are all positive, then $\mathbf{M}_V$ is also positive definite. The path integral from (A.5) is approximated on surface edge $e = e_{i,i+1}$ for functions g and h as follows

$$\int_e gh \approx \langle \mathbf{g}_e, \mathbf{h}_e \rangle_{SV,e} = \frac{\ell_{i,i+1}}{2} (\mathbf{g}_{2i+1} \mathbf{h}_{2i+1} + \mathbf{g}_{2i+2} \mathbf{h}_{2i+2}). \tag{A.13}$$

This discretization generates a diagonal positive definite matrix $\mathbf{M}_{SV}$.

Since ultimately we would like to mimic the integration by parts formula (A.7), it could be useful to extend the discrete divergence operator to mimic the extended continuous divergence operator (A.6). Following in this manner, it may be possible to construct a mimetic framework including a discrete divergence and gradient operators that maintain duality and extend to the boundary. However, we will refrain from doing so because at this point we have sufficient machinery to discretize the Poisson problem with second order convergence.

We are now ready to discretize our weak BVP (A.3). We will assume pure Neumann boundary conditions, so the boundary integral in (A.3) covers all of $\partial\Omega$. Substituting the discrete inner products (A.11) and (3.5) into (A.9) forms the discrete BVP: Find $\mathbf{p} \in \mathbb{R}^{N_V}$ such that

$$\mathbf{q}^\top \mathbf{G}^\top \mathbf{M}_E \mathbf{G} \mathbf{p} = \mathbf{q}^\top \mathbf{M}_V \mathbf{f} + \mathbf{q}^\top \tilde{\mathbf{M}}_{SV} \mathbf{g}^N \qquad \forall \mathbf{q} \in \mathbb{R}^{N_V},$$

where $\mathbf{f}$ samples the function f at each mesh node, $\mathbf{g}^N$ samples the function g^N on mesh nodes on $\partial\Omega$ as usual, and $\tilde{\mathbf{M}}$ is an $\mathbb{R}^{N_V} \times \mathbb{R}^{N_{SV}}$ extension of $\mathbf{M}_{SV}$:

$$\tilde{\mathbf{M}}_{SV} = \begin{pmatrix} \mathbf{0} \\ \mathbf{M}_{SV} \end{pmatrix}$$

defined for convenience. Furthermore, since $\mathbf{q}$ is arbitrary, we can reduce this to

$$\mathbf{G}^\top \mathbf{M}_E \mathbf{G} \mathbf{p} = \mathbf{M}_V \mathbf{f} + \tilde{\mathbf{M}}_{SV} \mathbf{g}^N.$$

This is precisely the discretization used in Example 6.1.4, which demonstrates second order convergence. For completeness we will derive the weak formulation for our specific BVP (A.1). The solutions to the weak form of (A.1) must satisfy

$$\int_\Omega q\nabla \cdot (\vec{u}^{**} - \Delta t \nabla p) = 0 \qquad \forall q \in H_0^1.$$

writing this as

$$\int_\Omega q\Delta t \nabla \cdot \nabla p = \int_\Omega q \nabla \cdot \vec{u}^{**} \qquad \forall q \in H_0^1,$$

and using the integration by parts formula (A.7) on both sides allows us to express this integral in terms we know how to discretize:

$$\int_{\partial\Omega} q(\hat{n} \cdot \Delta t \nabla p) - \Delta t \int_\Omega \nabla q \cdot \nabla p = \int_{\partial\Omega} q(\hat{n} \cdot \vec{u}^{**}) - \int_\Omega \nabla q \cdot \vec{u}^{**} \qquad \forall q \in H_0^1.$$

Substituting in Neumann boundary conditions $\hat{n} \cdot \Delta t \nabla p = \hat{n} \cdot \vec{u}^{**} - f^N$, we get

$$\int_{\partial\Omega} q(\hat{n} \cdot \vec{u}^{**}) - \int_{\partial\Omega} q f^N - \int_\Omega \nabla q \cdot \nabla p = \int_{\partial\Omega} q(\hat{n} \cdot \vec{u}^{**}) - \int_\Omega \nabla q \cdot \vec{u}^{**} \qquad \forall q \in H_0^1,$$

which reduces to

$$\int_\Omega \nabla q \cdot \nabla p = \int_\Omega \nabla q \cdot \vec{u}^{**} - \int_{\partial\Omega} q f^N \qquad \forall q \in H_0^1.$$

We know know how to discretize the boundary integral from (A.13) and the interior integral using (3.5) as usual:

$$\mathbf{q}^\top \Delta t \mathbf{G}^\top \mathbf{M}_E \mathbf{G} \mathbf{p} = \mathbf{q}^\top \mathbf{G}^\top \mathbf{M}_E \mathbf{u}^{**} - \mathbf{q}_\partial^\top \mathbf{M}_{SV} \mathbf{f}^N \tag{A.14}$$

for all $\mathbf{q} \in \mathbb{R}^{N_V}$ and $\mathbf{q}_\partial \in \mathbb{R}^{N_{SV}}$ where $\mathbf{q}_\partial$ is zero for all edges not subject to Neumann boundary conditions (this follows from the definition of H_0^1). The vector $\mathbf{q}_\partial$ comes from the extension of $\mathbf{q} \in \mathbb{R}^{N_V}$ to the boundary as described in the beginning of this section. The restriction on $\mathbf{q}_\partial$ can be encoded into the matrix $\tilde{\mathbf{M}}_{SV}$, which can be written as $\mathbf{L}$ defined in (3.27). Furthermore, f^N is defined by a prescribed boundary velocity normal to the liquid-solid boundary as in (2.5b). Given a velocity field $\vec{v}$ on the surface of the liquid, discretized by $\mathbf{v} \in \mathbb{R}^{2N_{SV}}$ using Notation 3.2.1, we can write $\mathbf{f}^N = \mathbf{N}\mathbf{v}$ where $\mathbf{N}$ is defined as in (3.28). Thus we can write (A.14) as

$$\mathbf{q}^\top \Delta t \mathbf{G}^\top \mathbf{M}_E \mathbf{G} \mathbf{p} = \mathbf{q}^\top \mathbf{G}^\top \mathbf{M}_E \mathbf{u}^{**} - \mathbf{q}^\top \mathbf{L}\mathbf{N}\mathbf{v}$$

where $\mathbf{q}$ is arbitrary, which further yields

$$\Delta t \mathbf{G}^\top \mathbf{M}_E \mathbf{G} \mathbf{p} = \mathbf{G}^\top \mathbf{M}_E \mathbf{u}^{**} - \mathbf{L}\mathbf{N}\mathbf{v}.$$

This is precisely the equation in (3.26). We demonstrate second order convergence of this method on irregular boundaries in Section 6.2.

Our discretization of the Poisson problem is not unlike the linear Galerkin method for simplicial meshes. In fact, the MFD method has been unified with higher order discretizations in a Galerkin framework called the *virtual element method* [Beirão da Veiga et al., 2013].

Appendix B

Code to Compute the Energy Hessian

We use a symbolic computation package [SymPy Development Team, 2014] to compute the energy Hessian. This is done in the Julia programming language [Bezanson et al., 2012] with a convenience wrapper for SymPy [Verzani, 2015]. Note that for simplicity we exclude the constant factor γ from the computations since it plays no role in the derivatives.

```
using SymPy
using LaTeXStrings
xi = symbols("x_i", real=true)
yi = symbols("y_i", real=true)
xip1 = symbols("x_{i+1}", real=true)
yip1 = symbols("y_{i+1}", real=true)
xim1 = symbols("x_{i-1}", real=true)
yim1 = symbols("y_{i-1}", real=true)

# Definition of surface area gradient (computed by hand) at node i
lp = sqrt((xi-xip1)^2+(yi-yip1)^2) # length of next edge
lm = sqrt((xi-xim1)^2+(yi-yim1)^2) # length of previous edge
fix = (xip1-xi)/lp - (xi-xim1)/lm # force x component
fiy = (yip1-yi)/lp - (yi-yim1)/lm # force y component

# Compute Hessian Stencil
Dixx = simplify(diff(fix,xi))
Dixy = simplify(diff(fiy,xi))

Diyx = simplify(diff(fix,yi)) # expect to be the same as Dixy
```

```
Diyy = simplify(diff(fiy,yi))

Dip1xx = simplify(diff(fix,xip1))
Dip1xy = simplify(diff(fiy,xip1))

Dip1yx = simplify(diff(fix,yip1)) # expect to be the same as Dip1xy
Dip1yy = simplify(diff(fiy,yip1))

assert(Diyx == Dixy)
assert(Dip1xy == Dip1yx)

# Prepare results for display
D(i,x,y) = "\\frac{\\partial f_{$(i)}^$(x)}{\\partial $(y)_{$(i)}}"
res = [
    "\$ $(D("i","x","x"))=$(SymPy.latex(Dixx))\$",
    "\$ $(D("i","x","y"))=$(D("i","y","x")) =  $(SymPy.latex(Dixy))\$",
    "\$ $(D("i","y","y"))=$(SymPy.latex(Diyy))\$",
    "\$ $(D("i+1","x","x"))=$(SymPy.latex(Dip1xx))\$",
    "\$ $(D("i+1","x","y"))=$(D("i+1","y","x")) =  $(SymPy.latex(Dip1xy))\$",
    "\$ $(D("i+1","y","y"))=$(SymPy.latex(Dip1yy))\$",
]

# Display results
for str in res
    display(LaTeXString(str))
end
```

This code generates the raw component-wise partial derivatives of the surface tension force needed to build the energy Hessian. For simplicity of exposition we simplify the output by expressing it in terms of edge vector components $\vec{e}_i = (e_i^x, e_i^y)$ with the following code:

```
# Express the above derivatives in terms of edge vector components
eix = symbols("{e^x_i}", real=true)
eiy = symbols("{e^y_i}", real=true)
ein = symbols("{\\|{e_{i}}\\|}", positive=true)
eim1x = symbols("{e^x_{i-1}}", real=true)
eim1y = symbols("{e^y_{i-1}}", real=true)
eim1n = symbols("{\\|{e_{i-1}}\\|}", positive=true)
function sim(f)
    f = subs(subs(subs(f,xip1-xi,eix),yip1-yi,eiy),eix^2+eiy^2,ein^2)
    f = subs(subs(subs(f,xi-xim1,eim1x),yi-yim1,eim1y),eim1x^2+eim1y^2,eim1n^2)
    simplify(f)
end
res = [
```

```
    "\$ $(D("i","x","x")) =  $(SymPy.latex(sim(Dixx)))\$",
    "\$ $(D("i","x","y")) = $(D("i","y","x")) =  $(SymPy.latex(sim(Dixy)))\$",
    "\$ $(D("i","y","y")) =  $(SymPy.latex(sim(Diyy)))\$",
    "\$ $(D("i+1","x","x")) =  $(SymPy.latex(sim(Dip1xx)))\$",
    "\$ $(D("i+1","x","y")) = $(D("i+1","y","x")) =  $(SymPy.latex(sim(Dip1xy)))\$",
    "\$ $(D("i+1","y","y")) =  $(SymPy.latex(sim(Dip1yy)))\$",
]
for str in res
    display(LaTeXString(str))
end
```

This code generates the following expressions for surface tension force derivatives:

$$\begin{aligned}
\frac{\partial f_i^x}{\partial x_i} &= -\frac{1}{\|\vec{e}_i\|} + \frac{{e_i^x}^2}{\|\vec{e}_i\|^3} - \frac{1}{\|\vec{e}_{i-1}\|} + \frac{{e_{i-1}^x}^2}{\|\vec{e}_{i-1}\|^3} \\
\frac{\partial f_i^x}{\partial y_i} = \frac{\partial f_i^y}{\partial x_i} &= \frac{e_i^x e_i^y}{\|\vec{e}_i\|^3} + \frac{e_{i-1}^x e_{i-1}^y}{\|\vec{e}_{i-1}\|^3} \\
\frac{\partial f_i^y}{\partial y_i} &= -\frac{1}{\|\vec{e}_i\|} + \frac{{e_i^y}^2}{\|\vec{e}_i\|^3} - \frac{1}{\|\vec{e}_{i-1}\|} + \frac{{e_{i-1}^y}^2}{\|\vec{e}_{i-1}\|^3} \\
\frac{\partial f_{i+1}^x}{\partial x_{i+1}} &= \frac{{e_i^y}^2}{\|\vec{e}_i\|^3} \\
\frac{\partial f_{i+1}^x}{\partial y_{i+1}} = \frac{\partial f_{i+1}^y}{\partial x_{i+1}} &= -\frac{e_i^x e_i^y}{\|\vec{e}_i\|^3} \\
\frac{\partial f_{i+1}^y}{\partial y_{i+1}} &= \frac{{e_i^x}^2}{\|\vec{e}_i\|^3}.
\end{aligned}$$

Note that due to the symmetry and sparsity of the Hessian as seen in (4.11), these are the only derivatives we need to compute to form the stencil in (4.12).

References

Alexa, M. and Wardetzky, M. (2011). Discrete laplacians on general polygonal meshes. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 102:1–102:10, New York, NY, USA. ACM.

Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M. (2005). Variational tetrahedral meshing. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 617–625, New York, NY, USA. ACM.

Autodesk (2015). Maya. http://autodesk.com/maya.

Batty, C. (in submission 2015). A cell-centred finite volume method for the poisson problem on non-graded quadtrees with second order accurate gradients.

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

Beirão da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L. D., and Russo, A. (2013). Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214.

Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145.

Brackbill, J., Kothe, D., and Zemach, C. (1992). A continuum method for modeling surface tension. *Journal of Computational Physics*, 100(2):335 – 354.

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, second edition.

Brochu, T. and Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493.

Bronson, J., Levine, J., and Whitaker, R. (2014). Lattice cleaving: A multimaterial tetrahedral meshing algorithm with guarantees. *Visualization and Computer Graphics, IEEE Transactions on*, 20(2):223–237.

Chentanez, N., Müller, M., Macklin, M., and Kim, T.-Y. (2015). Fast grid-free surface tracking. *ACM Trans. Graph.*, 34(4):148:1–148:11.

Chorin, A. J., Marsden, J. E., and Marsden, J. E. (1990). *A mathematical introduction to fluid mechanics*, volume 3. Springer.

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

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

de Gennes, P. G. (1985). Wetting: statics and dynamics. *Rev. Mod. Phys.*, 57:827–863.

Domelevo, K. and Omnes, P. (2005). A finite volume method for the laplace equation on almost arbitrary two-dimensional grids. *ESAIM: Mathematical Modelling and Numerical Analysis*, 39:1203–1249.

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

Enright, D., Nguyen, D., Gibou, F., and Fedkiw, R. (2003). Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *ASME/JSME 2003 4th Joint Fluids Summer Engineering Conference*, pages 337–342. American Society of Mechanical Engineers.

Eymard, R., Gallouët, T., and Herbin, R. (2007). A new finite volume scheme for anisotropic diffusion problems on general grids: convergence analysis. *Comptes Rendus Mathematique*, 344(6):403 – 406.

Ferstl, F., Ando, R., Wojtan, C., Westermann, R., and Thuerey, N. (2016). Narrow band FLIP for liquid simulations. *Computer Graphics Forum (Eurographics)*, 35(2):to appear.

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.

Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Halley, J. W. (2006). *Statistical Mechanics: From first principles to macroscopic phenomena*. Cambridge University Press.

Harlow, F. (1963). The particle-in-cell method for numerical solution of problems in fluid dynamics. Symposium on Appl.Math. XV. American Math. Soc.

Harlow, F. H., Welch, J. E., et al. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 8(12):2182.

Hong, J.-M. and Kim, C.-H. (2005). Discontinuous fluids. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 915–920, New York, NY, USA. ACM.

Hyman, J. and Shashkov, M. (1998). Approximation of boundary conditions for mimetic finite-difference methods. *Computers & Mathematics with Applications*, 36(5):79 – 99.

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

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., and Teschner, M. (2014b). SPH Fluids in Computer Graphics. In Lefebvre, S. and Spagnuolo, M., editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association.

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

Jiao, X. (2007). Face offsetting: A unified approach for explicit moving interfaces. *Journal of Computational Physics*, 220(2):612 – 625.

Johansen, H. and Colella, P. (1998). A cartesian grid embedded boundary method for poisson's equation on irregular domains. *Journal of Computational Physics*, 147(1):60 – 85.

Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J. (2007). Simulation of bubbles in foam with the volume control method. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H.-P. (2001). Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 57–66, New York, NY, USA. ACM.

Labelle, F. and Shewchuk, J. R. (2007). Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

Liovic, P., Francois, M., Rudman, M., and Manasseh, R. (2010). Efficient simulation of surface tension-dominated flows through enhanced interface geometry interrogation. *Journal of Computational Physics*, 229(19):7520 – 7544.

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

Losasso, F., Fedkiw, R., and Osher, S. (2005). Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35:995–1010.

Losasso, F., Gibou, F., and Fedkiw, R. (2004). Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 457–462, New York, NY, USA. ACM.

Misztal, M. K., Erleben, K., Bargteil, A., Fursund, J., Christensen, B. B., Bærentzen, J. A., and Bridson, R. (2012). Multiphase flow of immiscible fluids on unstructured moving meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, pages 97–106, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Müller, M. (2009). Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 237–245, New York, NY, USA. ACM.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

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.

Quan, S., Lou, J., and Schmidt, D. P. (2009). Modeling merging and breakup in the moving mesh interface tracking method for multiphase flow simulations. *Journal of Computational Physics*, 228(7):2660 – 2675.

Rocchini, C., Cignoni, P., Ganovelli, F., Montani, C., Pingi, P., and Scopigno, R. (2001). Marching intersections: an efficient resampling algorithm for surface management. In *Shape Modeling and Applications, SMI 2001 International Conference on.*, pages 296–305.

Schroeder, C., Zheng, W., and Fedkiw, R. (2012). Semi-implicit surface tension formulation with a lagrangian surface mesh on an eulerian simulation grid. *J. Comput. Phys.*, 231(4):2092–2115.

Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J. (2008). An unconditionally stable maccormack method. *J. Sci. Comput.*, 35(2-3):350–371.

Shewchuk, J. R. and Si, H. (2014). Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 290:290–290:299, New York, NY, USA. ACM.

Si, H. (2015). Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36.

SideFX (2015). Houdini 15. http://sidefx.com/houdini.

Solenthaler, B. and Pajarola, R. (2009). Predictive-corrective incompressible sph. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 40:1–40:6, New York, NY, USA. ACM.

Stam, J. (1999). Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Sussman, M. and Ohta, M. (2009). A stable and efficient method for treating surface tension in incompressible two-phase flow. *SIAM Journal on Scientific Computing*, 31(4):2447–2471.

SymPy Development Team (2014). *SymPy: Python library for symbolic mathematics.*

Thomas, J.-M. and Trujillo, D. (1999). Mixed finite volume methods. *International Journal for Numerical Methods in Engineering*, 46(9):1351–1366.

Thürey, N., Wojtan, C., Gross, M., and Turk, G. (2010). A multiscale approach to mesh-based surface tension flows. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 48:1–48:10, New York, NY, USA. ACM.

Veiga, L. B., Manzini, G., and Lipnikov, K. (2014). *The mimetic finite difference method for elliptic problems*. Springer.

Verzani, J. (2015). Julia interface to sympy via pycall.

Wang, H., Mucha, P. J., and Turk, G. (2005). Water drops on surfaces. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 921–929, New York, NY, USA. ACM.

Wojtan, C., Thürey, N., Gross, M., and Turk, G. (2009). Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 76:1–76:10, New York, NY, USA. ACM.

Zheng, W., Yong, J.-H., and Paul, J.-C. (2009). Simulation of bubbles. *Graphical Models*, 71(6):229 – 239. 2006 {ACM} SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006).

Zheng, W., Zhu, B., Kim, B., and Fedkiw, R. (2015). A new incompressibility discretization for a hybrid particle {MAC} grid representation with surface tension. *Journal of Computational Physics*, 280:96 – 142.

Zhu, Y. and Bridson, R. (2005). Animating sand as a fluid. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 965–972, New York, NY, USA. ACM.