A Level Set and Sharp Interface Approach for Simulating Incompressible Two-Phase Flow

by

Michael Fattori

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisors: Prof. Justin Wan,
Prof. Gladimir Baranoski

Waterloo, Ontario, Canada, 2014

© Michael Fattori 2014

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

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

Abstract

In this paper we describe an algorithm to simulate *two-phase* fluid flow. Two-phase flow refers to the interaction of two fluid mediums which do not mix, such as water and air. One of the main concerns in two-phase flow simulations is how to represent and evolve the interface between the two fluids. We elect to *implicitly* represent the interface as all points $\vec{x}$ where a scalar function ϕ is equal to zero. Such *level set methods* of fluid interface representation can be seen in the related literature [3, 4, 5, 7, 17, 18, 19, 20]. We largely discretize the equations of motion (the Navier-Stokes equations) using the discretizations in [8]. At the interface between the fluids, we employ ideas from the *ghost fluid method* to preserve the sharp change in density and viscosity when solving for pressure and discretizing the *stress tensor*.

Our description of the algorithm is detailed enough that readers should be able to build their own implementations. We use our implementation to simulate a water drop falling through air into a pool of water below. The parameters of the liquid (water) phase are individually explored and results are presented in Chapter 5. We provide a discussion of the technical issues of our algorithm and suggest improvements. Directions for future work are also discussed.

Acknowledgements

I would like to thank my supervisors, Justin Wan and Gladimir Baranoski. Thank you for giving me the chance to work with you, and the invaluable support you provided, both in my undergraduate and graduate career.

I also want to thank my parents, Anne and Mark Fattori. Thank you for getting me here, supporting me, and loving me.

I would like to thank my second reader, Serge D'Alessio, for providing excellent notes and helping me polish this work.

A special thanks to Christopher Batty, for directing me to excellent resources, and helping me understand crucial aspects of the algorithm.

Finally, a big thanks to my friends in the Computational Math program. We made it!

Dedication

For my brother, Aaron, who might actually read this thing.

Table of Contents

List of Tables

List of Figures

List of Acronyms

List of Symbols

ϕ	The *level set function* ϕ implicitly defines the interface between the two fluid phases as all points $\vec{x}$ where $\phi(\vec{x}) = 0$.	2
∇	Gradient operator	3
$\nabla\cdot$	Divergence operator	3
p	Pressure	3
ρ	Density	3
μ	Viscosity	3
$\vec{u}$	Fluid velocity vector	4
$\vec{g}$	External force vector acting on the fluid.	4
$\boldsymbol{\sigma}$	Stress Tensor	4
$\mathbf{I}$	Identity Matrix	5
λ	Thermodynamic material constant	5
$\boldsymbol{\delta}$	Strain tensor	5
L	Characteristic length	5
u_∞	Characteristic speed	5
p_∞	Characteristic pressure	5
u	x-component of velocity	6
v	y-component of velocity	6
$\vec{n}$	Unit normal vector to boundary wall.	7
$\mathbb{R}$	Set of all real numbers.	9
Ω	Fluid domain	10
Γ_k	The k level set of the level set function ϕ	10
Γ	Zero level set of level set function ϕ	10
Ω_+	All points $\vec{x} \in \Omega$ for which $phi(\vec{x}) > 0$.	10
Ω_-	All points $\vec{x} \in \Omega$ for which $phi(\vec{x}) \leq 0$.	10
T	Final simulation time.	12
Δx	Grid spacing in the x direction.	12
S	Sign function.	15

S_ϵ	Smoothed sign function.	16
ϵ	Small number used in level set reinitialization.	16
g_x	External force in the x direction.	18
g_y	External force in the y direction.	18
Δy	Grid Spacing in the y direction	18
$c_{i,j}$	Grid cell corresponding to indices i and j.	18
n_x	Number of physical cells in x direction	18
n_y	Number of physical cells in y direction	18
γ	A parameter controlling the balance of centered and *donor-cell* discretizations of the convective terms in the Navier-Stokes equations.	22
Re	The Reynolds number.	24
σ	Surface tension coefficient, controlling the strength of surface tension.	24
τ	A safety factor for the time step Δt. We set Δt to its least upper bound required for stability multiplied by τ.	25
R	Radius of water drop in our simulations.	25
h	Height of the pool below our water drop.	25
F	A variable composed of velocity terms in the momentum equation for u, defined for convenience.	27
G	A variable composed of velocity terms in the momentum equation for v, defined for convenience.	27
κ	Curvature of level set function ϕ, used to defined curvature of the fluid interface Γ	28
θ	Parameter between zero and one indicating the location of the interface between two grid points.	28
D^0_i	Zeroth divided difference of a function ϕ at the center of cell i.	51
$D^1_{i+\frac{1}{2}}$	First divided difference of a function ϕ midway between the centers of cells i and $i+1$.	51
D^2_i	Second divided difference of a function ϕ at cell center i.	51
$D^3_{i+\frac{1}{2}}$	First divided difference of a function ϕ midway between the centers of cells i and $i+1$.	51

Chapter 1

Introduction

The first papers in computational fluid dynamics (CFD) were published in 1965 [9, 10], and since then a variety of techniques have been developed. Each technique will typically look to solve the *Navier-Stokes* equations, either directly or indirectly. Particle based methods, such as Lattice-Boltzmann or Smooth Particle Hydrodynamics (SPH), can be considered indirect methods. They simulate a number of particles which interact with each other and whose cumulative behaviour can be shown to conform to the motion described by the Navier-Stokes equations. When solved directly, the Navier-Stokes equations are treated as partial differential equations. A variety of techniques exist to solve partial differential equations such as finite difference, finite volume, finite element and spectral methods. Each of these methods have been used to solve the Navier-Stokes equations and, hence, to simulate fluid flows.

We are often interested in fluids that have free boundaries, as opposed to static physical boundaries like solid walls. An example of such a *free-boundary problem* is a pitcher of water being poured into a glass. The space that the fluid occupies continually changes, and the boundary of that space moves freely. This introduces some complexity to the simulation since the fluid domain is treated differently than the rest of the domain, but also changes with time. Creating an algorithm to account for the changing fluid domain can be a challenge. There are two major ways to represent the boundary between the fluids: *explicitly* and *implicitly*.

Boundaries can be explicitly represented with line segments in two dimensions, or triangles in three dimensions. In these simulations, the boundary must be updated at each time step. During merging or splitting of fluid bodies, the line segments or triangles must be connected or split to properly account for the new fluid domain [16], which can be difficult to program.

Using *level set* functions, we can implicitly represent the fluid boundary. This is done by associating the fluid boundary with all points where a scalar function ϕ is equal to zero. These methods implicitly deal with the merging and splitting of fluid bodies. Hence, they do not require the programmer to "handle" these situations, as in explicit methods.

In this essay, we look to create a free-boundary fluid simulation using a finite difference method and level set boundary representation. Free-boundary simulations sometimes treat the region surrounding the fluid domain as empty space as in [8]. These simulations can be very useful, but do not represent some of our most common experiences with fluid. We often interact with fluid that is surrounded by air as in the example of water poured from a pitcher to a glass. In that example, the water must push through the air which offers some resistance. When the water reaches the glass, we also expect bubbles to form since the air does not immediately yield its physical space to the water. The term used to reference two interacting fluids which do not mix is *two-phase flow*. Our simulations will be of two-phase flow.

1.1 Finite Difference Two-Phase Flow Techniques

A discussion of all the techniques in CFD would be outside the scope of this essay. We are focusing here on two-phase flow simulations which use finite difference methods to solve the Navier-Stokes equations. In the context of finite difference methods, two-phase flow can be simulated in a manner very similar to single-phase flow. The only difference is how to treat the interface between the two phases. In finite difference methods, grid points store parameters found in the equations of motion. Each grid point is updated according to its current value and the values of its neighbouring grid points. If the interface between the phases is far from a grid point, that point can be updated in the same manner as for single-phase flow. Grid points near the interface have to be handled more carefully.

The main problem at the interface is that the fluid parameters, density and viscosity, change abruptly. In finite differences, this sharp change at the interface has been handled in two major ways:

- Preserving the sharp interface using the *ghost fluid method* as in [7].
- Smoothing the sharp interface in a way amenable to finite difference approximations as in [17, 19].

We will be using ideas from the ghost fluid method in our simulations. One of the main features of the ghost fluid method is how it treats the change in pressure and viscosity at the fluid interface. In most finite difference methods [3, 7, 8, 17, 19] the pressure for the upcoming time step is solved for implicitly using the current fluid velocity. The pressure is solved for in the *variable coefficient Poisson equation*,

$$\nabla \cdot \left(\frac{1}{\rho} \nabla p \right) = b,$$

where p is pressure, ρ is density, and b is formed using velocities from the current time step. Away from the interface, ρ can simply be set to the density of the surrounding fluid. Near the interface, ρ is specially chosen between the two fluid densities in accordance with the ghost fluid technique. We can also add surface tension by modifying the right hand side b. We discuss how ρ is chosen in Section 4.2.2. To account for the change in the viscosity μ at the interface, we will employ similar ideas.

We will begin our discussion by introducing the Navier-Stokes equations in Chapter 2. In order to track the interface between the fluids, we will use a level set function ϕ, and so we will discuss level set methods in Chapter 3. In Chapter 4, we discuss discretization of the Navier-Stokes equations, and end with a summary of our entire simulation algorithm. The goal is to provide enough detail in Chapter 4 that any interested readers can replicate our results.

We will present our results in Chapter 5. In Chapter 6, we conclude with a brief discussion of the strengths and weaknesses of our algorithm, and outline directions for future work.

Chapter 2

Equations of Motion

In general, *laminar* flows (flows without *turbulence*) are modelled with the following form of the well known *Navier-Stokes equations*, [8]

$$\frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \vec{u}) = 0, \tag{2.1}$$

and

$$\frac{\partial}{\partial t}(\rho \vec{u}) + (\vec{u} \cdot \nabla)(\rho \vec{u}) + (\rho \vec{u})\nabla \cdot \vec{u} - \rho \vec{g} - \nabla \cdot \boldsymbol{\sigma} = 0, \tag{2.2}$$

where $\vec{u}$ is the fluid velocity, p is the pressure, ρ is the density of the fluid, $\vec{g}$ represents external forces (*e.g.*, gravity) and $\boldsymbol{\sigma}$ is the **stress tensor** which represents the internal friction forces amongst the fluid particles. Equation (2.1) is called the *continuity equation* and comes from imposing conservation of mass onto the fluid. Equation (2.2) is called the *momentum equation* derived from imposing conservation of momentum onto the fluid. Both fluid phases will be modelled with these equations after the simplifications below.

We are interested in modelling *viscous* fluids. Viscosity is a measure of a fluid's resistance to deformation by shear stress or tensile stress. Informally, it corresponds to the notion of the "thickness" of a fluid, and it is a result of frictional forces between the particles. Hence we must incorporate internal friction into our model (which is unnecessary for inviscid gases). The flows described by these equations depend heavily on the stress tensor $\boldsymbol{\sigma}$. For *Newtonian* fluids obeying the *Stokes assumption* the stress tensor can be modelled as [8]:

$$\boldsymbol{\sigma} := -p\mathbf{I} + \boldsymbol{\tau} := (-p + \lambda \nabla \cdot \vec{u})\mathbf{I} + 2\mu\boldsymbol{\delta}.$$

Here we have $\boldsymbol{\tau} = (\lambda \nabla \cdot \vec{u})\mathbf{I} + 2\mu\boldsymbol{\delta}$. The parameters μ and λ are thermodynamic material constants and $\boldsymbol{\delta}$ is the *strain tensor*,

$$\boldsymbol{\delta} := \frac{1}{2}\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right]_{i,j=1,2}.$$

Substituting this version of the stress tensor into Equation (2.2) gives us

$$\frac{\partial}{\partial t}(\rho\vec{u}) + (\vec{u}\cdot\nabla)(\rho\vec{u}) + (\rho\vec{u})\nabla\cdot\vec{u} - \rho\vec{g} - \nabla\cdot((-p + \lambda\nabla\cdot\vec{u})\mathbf{I} + 2\mu\boldsymbol{\delta}) = 0,$$

which, can be simplified to

$$\frac{\partial}{\partial t}(\rho\vec{u}) + (\vec{u}\cdot\nabla)(\rho\vec{u}) + (\rho\vec{u})\nabla\cdot\vec{u} - \rho\vec{g} + \nabla p - \lambda\nabla(\nabla\cdot\vec{u}) - \nabla\cdot(2\mu\boldsymbol{\delta}) = 0.$$

Under the assumption of incompressibility ($\rho(\vec{x},t) = \rho_\infty =$ constant), the continuity equation simplifies to

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

This allows us to simplify the momentum equation further to

$$\frac{\partial}{\partial t}(\rho\vec{u}) + (\vec{u}\cdot\nabla)(\rho\vec{u}) - \rho\vec{g} + \nabla p - \nabla\cdot(2\mu\boldsymbol{\delta}) = 0.$$

Again, using the incompressibility condition ($\rho = \rho_\infty$), we can divide the whole equation by ρ which gives,

$$\frac{\partial}{\partial t}\vec{u} + (\vec{u}\cdot\nabla)\vec{u} + \frac{1}{\rho}\nabla\cdot p = \frac{1}{\rho}\nabla\cdot(2\mu\boldsymbol{\delta}) + \vec{g}.$$

We nondimensionalize this equation as in [8] by introducing dimensionless variables,

$$\vec{x}^* := \frac{\vec{x}}{L}, \quad t^* := \frac{u_\infty t}{L}, \quad \vec{u}^* := \frac{\vec{u}}{u_\infty}, \quad \vec{g}^* = \frac{L}{u_\infty^2}\vec{g},$$

and scaled pressure,

$$p^* := \frac{p - p_\infty}{u_\infty^2},$$

which gives us,

$$\frac{\partial}{\partial t^*}\vec{u}^* + (\vec{u}^*\cdot\nabla^*)\vec{u}^* + \frac{1}{\rho}\nabla^*\cdot p^* = \frac{1}{\rho u_\infty L}\nabla^*\cdot(2\mu\boldsymbol{\delta}) + \vec{g}^*.$$

Here we have introduced characteristic length L, characteristic speed u_∞, and characteristic pressure p_∞. Typically, pressure is also nondimensionalized by dividing p^* by the density ρ. We keep the density out of our scaled pressure since density changes at the interface between fluid phases. We need to take this density change into account when we define our discrete pressure derivatives across the fluid interface in Section 4.2.2. Similarly, we leave μ inside the stress tensor term to motivate our stress tensor discretizations in Section 4.2.3.

The divergence and gradient operators in the previous equation are with respect to the variable $\vec{x}^*$ instead of $\vec{x}$. Dropping the * notation, we get the final version of the momentum equation:

$$\frac{\partial}{\partial t}\vec{u} + (\vec{u}\cdot\nabla)\vec{u} + \frac{1}{\rho}\nabla\cdot p = \frac{1}{\rho u_\infty L}\nabla\cdot(2\mu\boldsymbol{\delta}) + \vec{g}. \tag{2.4}$$

If we had removed the viscosity μ from the stress tensor term, we would see the reciprocal of the *Reynolds number*, $Re = \rho u_\infty L/\mu$, as our coefficient on the right hand side of Equation (2.4). Since ρ and μ are parameters of the fluid phases, we modify the Reynolds number by modifying the product $u_\infty L$. This is explored in Section 5.5.

So we model *laminar* flows of *viscous*, *incompressible fluids* using Equations (2.3) and (2.4) [8]. Before moving on we will put these equations into component form in preparation for the discretization in Chapter 4.

We let $\vec{u} = \begin{pmatrix} u \\ v \end{pmatrix}$. We will deal with the $\nabla\cdot(2\mu\boldsymbol{\delta})$ term first. We have

$$\begin{aligned}
\nabla\cdot(2\mu\boldsymbol{\delta}) &= \nabla\cdot\left(2\mu\frac{1}{2}\begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} & 2\frac{\partial v}{\partial y}\end{pmatrix}\right) \\
&= \begin{pmatrix} 2\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right) \\ \frac{\partial}{\partial x}\left(\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right) + 2\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right) \\ \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\end{pmatrix} + \mu\begin{pmatrix} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x\partial y} \\ \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 u}{\partial y\partial x}\end{pmatrix}
\end{aligned}$$

$$= \begin{pmatrix} \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right) \\ \\ \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) \end{pmatrix} + \mu \begin{pmatrix} \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \\ \\ \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right) \\ \\ \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) \end{pmatrix} + \mu\nabla\left(\nabla\cdot\vec{u}\right).$$

From the continuity equation we have $\nabla\cdot\vec{u} = 0$. Hence the second term on the right hand side of the expression above disappears, and we get

$$\nabla\cdot(2\mu\boldsymbol{\delta}) = \begin{pmatrix} \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right) \\ \\ \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) \end{pmatrix}.$$

We can now write the momentum Equation (2.4) in component form:

$$\frac{\partial u}{\partial t} + \frac{\partial\left(u^2\right)}{\partial x} + \frac{\partial\left(uv\right)}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} = \frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\right) + g_x, \tag{2.5}$$

$$\frac{\partial v}{\partial t} + \frac{\partial\left(uv\right)}{\partial x} + \frac{\partial\left(v^2\right)}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} = \frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\right) + g_y. \tag{2.6}$$

Now we have the equations of motion within the computational domain, and we just need to consider the boundary conditions. Our simulation assumes solid wall boundaries with the *free-slip* condition, as in [19]. This requires that the velocity component normal to the boundary is zero, and also that the normal derivative of the velocity is zero at the boundary. This leads to the two conditions,

$$\vec{u}\cdot\vec{n} = 0, \quad \frac{\partial\vec{u}}{\partial\vec{n}} = 0,$$

where $\vec{n}$ is the unit normal to the boundary wall, and $\frac{\partial\vec{u}}{\partial\vec{n}}$ is the directional derivative of the velocity $\vec{u}$ in the direction of $\vec{n}$. This allows the simulated fluid to slide along the wall without passing through it.

We can now move on to Chapter 3 to discuss the use of level sets to track the interface between the two fluids. In Chapter 4 we will revisit the fluid equations and introduce their discretized forms.

Chapter 3

Interface Tracking with Level Sets

In our two-phase fluid simulation we make use of level sets to track the interface between the two fluids (*e.g.*, water and air). We will discuss the problem of surface representation in the following subsection, then move on to the concept of level sets.

3.1 Surface Representation

In our two-phase flow simulations we need to represent the interface between the two phases. For air and water the interface would be the surface of the water, hence the term "surface representation". In two dimensions the surface can be represented as a curve or multiple curves. There are several ways one can represent a curve mathematically.

- *Functional representation*: The curve is the plot of $y = f(x)$, where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function.
- *Parametric representation*: The curve is defined as all points $\begin{pmatrix} x(s) \\ y(s) \end{pmatrix}$ for $s \in [a, b] \subset \mathbb{R}$, where x and y are continuous functions of the parameter s.
- *Vertices and edges*: The curve is a series of connected line segments, represented as a series of vertices and edges connecting them.
- *Implicit representation*: The curve is represented as all points $\vec{x} \in \mathbb{R}^2$ satisfying $\phi(\vec{x}) = 0$ for some function $\phi : \mathbb{R}^2 \to \mathbb{R}$.

With functional representation we cannot represent a curve which does not pass the vertical line test, such as a circle. Parametric representation is more robust but choosing functions x and y to represent a general curve can be difficult, and it is unclear how one would evolve the curve in time to correspond to the fluid flow.

Vertices and edges are robust since we can approximate any curve as a series of line segments. One can even evolve the curve in time by pushing vertices with the underlying fluid velocity as in [16]. However, as vertices drift apart it can become necessary to add more vertices to keep the approximation to the curve accurate. Another difficulty arises when two fluid bodies must merge, requiring us to "surgically" remove vertices and change connecting edges, as well as to identify the scenario in the first place. A similar difficulty arises when a fluid body splits in two.

Luckily, the implicit representation is very robust and useful for fluid interface tracking. Not only can we evolve the function ϕ using the fluid velocity, but we can define an interior and exterior to the curve $\phi(x, y) = 0$ using the sign of ϕ, which in turn can distinguish between the two fluid regions. Since the curve is all points where ϕ is a constant value, we also call this a *level set representation* of the curve. The remaining sections of this chapter discuss level sets more formally, and introduce the relevant tools we will need to work with them.

3.2 Level Sets

Suppose we have a function $\phi : \Omega \to \mathbb{R}$, with $\Omega \subseteq \mathbb{R}^n$. Then the k level set of ϕ, denoted by Γ_k, is defined as,

$$\Gamma_k := \{\vec{x} \in \Omega | \phi(\vec{x}) = k\}.$$

Since we can always cast the k level set of ϕ as the zero level set of $\phi - k$, we typically restrict ourselves to discussing the zero level set of a function ϕ. We will correspondingly denote Γ_0 as simply Γ. Typically a level set of a function ϕ whose domain is $\Omega \subseteq \mathbb{R}^n$ is an $(n-1)$-dimensional surface in Ω. As an example, we can describe the surface of a sphere with radius 1 centered at the origin as the zero level set of the function $\phi(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1$.

The function ϕ also implicitly separates Ω into two regions Ω_+ and Ω_- defined simply as,

$$\Omega_+ := \{\vec{x} \in \Omega | \phi(\vec{x}) > 0\},$$

and

$$\Omega_- := \{\vec{x} \in \Omega | \phi(\vec{x}) < 0\}.$$

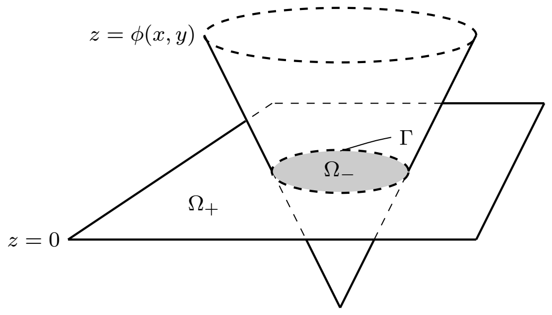

Figure 3.1: In this image the function $\phi(x, y) = \sqrt{x^2 + y^2} - 1$ defines a cone $z = \phi(x, y)$ whose intersection with the plane $z = 0$ is the unit circle. Points (x, y) interior to the unit circle have the property $\phi(x, y) < 0$, and so the interior of the unit circle defines Ω_- (in grey). Points (x, y) exterior to the unit circle have the property $\phi(x, y) > 0$, so the exterior of the unit circle defines Ω_+ (the white portion of the plane $z = 0$). The unit circle itself defines the zero level set Γ.

The level set Γ separates Ω_+ and Ω_-, and we can easily check which region a point $\vec{x}$ belongs to by checking the sign of $\phi(\vec{x})$. We often refer to Ω_- as the "inside" or "interior" region and Ω_+ as the "outside" or "exterior" region. Figure 3.1, should help illustrate the concept.

In many applications, we want Γ to belong to either Ω_+ or Ω_-. For instance, in our fluid solver, we are going to use Ω_+ to define the space occupied by air and Ω_- to define the space occupied by water. If we include Γ in neither Ω_+ nor Ω_-, then all points $\vec{x}$ with $\phi(\vec{x}) = 0$ are neither air nor water, which makes no physical sense. By convention, we usually redefine Ω_- as

$$\Omega_- := \{\vec{x} \in \Omega | \phi(\vec{x}) \leq 0\},$$

so that $\Gamma \subset \Omega_-$.

While this is a fine way to define a surface or a curve, we need some way of controlling how the surface moves in time. Fortunately, we can control how this surface moves using the *level set equation* [6]:

$$\phi_t + F|\nabla\phi| = 0,$$

where ϕ_t is the time derivative of ϕ. This PDE describes a function $\phi(\vec{x}, t)$ whose level set moves with speed F in the direction of the outward normal to the level set. F does not need to be a constant, often depending on the curvature of the level set, or on factors in the environment of the level set.

A major benefit of evolving a surface (or curve) using the level set equation is that the surface itself is never tracked explicitly. Because of this, when a surface splits or merges with another surface, there is no need to "handle" the situation for the programmer. All of the complications are handled implicitly, and it is simply a matter of being able to identify the zero level set of a function in general.

3.3 Level Set Motion Under an Externally Generated Velocity Field

We will be using a slightly modified version of the level set equation. We want to track the boundary between our two fluids using the level set of a function $\phi(x, y)$, and we want to "push" the level set using the underlying fluid velocity field. In general, if we have an externally generated velocity field $\vec{u}(\vec{x}, t) = \begin{pmatrix} u(\vec{x}, t) \\ v(\vec{x}, t) \end{pmatrix}$, we can push the level set in the direction of $\vec{u}$ by solving the following PDE [6]:

$$\phi_t + \vec{u} \cdot \nabla\phi = 0. \tag{3.1}$$

This equation is just a simple advection equation, and is perhaps easier to work with in the following form:

$$\phi_t + u\phi_x + v\phi_y = 0. \tag{3.2}$$

Here ϕ_x is the derivative of ϕ in the x direction, and ϕ_y is the derivative in the y direction.

We can solve this advection equation using an upwinding scheme. To illustrate the use of upwinding, we can work with the one dimensional advection equation and extend to two dimensions afterwards. The one dimensional advection equation is

$$\phi_t + a\phi_x = 0.$$

Suppose our spatial domain is $x \in \mathbb{R}$[1], and the time domain is $t \in [0, T]$. In finite difference schemes, we discretize time and space. We can discretize space using the points $x_i := i\Delta x$ for some small grid size Δx. For explicit time stepping schemes, such as forward Euler, we typically take timesteps of size Δt which satisfy the CFL condition $\Delta t \leq \dfrac{\Delta x}{a}$. Beginning with time $t_0 = 0$ we can discretize time by considering our problem only at times $t_n = n\Delta t$, where Δt is the time step. We will denote spatial indices with a subscript and time indices with a superscript, so that $\phi_i^n := \phi(x_i, t_n)$.

[1]Note that if we had a bounded domain such as $x \in [a, b]$ then we could choose the $N + 1$ discretization points $x_i := a + i\Delta x, i = 0, \dots, N$ with $\Delta x := \dfrac{b - a}{N}$.

We need approximations to the derivative terms in Equation (3.2). Using a forward Euler time discretization, our differential equation becomes the corresponding difference equation:

$$\phi_i^{n+1} = \phi_i^n - \Delta t \left(a\phi_x \right).$$

We only have to decide on a finite difference approximation of ϕ_x to formalize our method. The upwinding scheme suggests that when $a > 0$ we approximate $\phi_x(x_i, t_n)$ with

$$\left(\phi_x^-\right)_i^n = \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x},$$

and when $a < 0$ we approximate $\phi_x(x_i, t_n)$ with

$$\left(\phi_x^+\right)_i^n = \frac{\phi_{i+1}^n - \phi_i^n}{\Delta x}.$$

When $a = 0$ either choice is equivalent since we have $a\phi_x = 0$. In this case, purely by convention, we will choose the approximation $\left(\phi^+\right)_i^n$. For a more thorough discussion of the motivations behind the upwinding scheme readers can look in Chapter 3 of [6].

Upwinding as presented above is first order accurate, but the accuracy can be improved by using higher order approximations to ϕ_x. In particular, we can use third order accurate ENO approximations, which we will cover in more detail in Appendix A.

The extension to two dimensions is straightforward. The spatial ϕ derivatives in Equation (3.2) can be considered separately. We will have discretized the second space dimension in the same manner as the first, so that we have discrete points (x_i, y_j). The spatial derivatives are then approximated as

$$\left(\phi_x\right)_{i,j} = \begin{cases} \dfrac{\phi_{i,j}^n - \phi_{i-1,j}^n}{\Delta x}, & \text{if } u_{i,j} > 0, \\ \dfrac{\phi_{i+1,j}^n - \phi_{i,j}^n}{\Delta x}, & \text{if } u_{i,j} \leq 0, \end{cases}$$

and

$$\left(\phi_y\right)_{i,j} = \begin{cases} \dfrac{\phi_{i,j}^n - \phi_{i,j-1}^n}{\Delta y}, & \text{if } v_{i,j} > 0, \\ \dfrac{\phi_{i,j+1}^n - \phi_{i,j}^n}{\Delta y}, & \text{if } v_{i,j} \leq 0. \end{cases}$$

For one dimension, we discussed a constant wave speed a, but in two dimensions we now use wave speeds u and v from $\vec{u}(\vec{x}, t) = \begin{pmatrix} u(\vec{x}, t) \\ v(\vec{x}, t) \end{pmatrix}$ which are not constant. The time stepping algorithm still works as described above but the CFL condition now becomes,

$$\Delta t \leq \frac{\Delta x}{\max\limits_{i,j}\{|u_{i,j}|, |v_{i,j}|\}},$$

where the max of $|u_{i,j}|$ and $|v_{i,j}|$ is taken over the entire computational domain.

3.3.1 Level Set Reinitialization to Signed Distance Function

Let Γ be the level set of a function $\phi : \Omega \to \mathbb{R}$, with $\Omega \subseteq \mathbb{R}^n$. Then the corresponding signed distance function ϕ_S satisfies

$$\phi_S(\vec{x}) = \min_{\vec{y}\in\Gamma}\{|\vec{x} - \vec{y}|\} \cdot \operatorname{sgn}(\phi(\vec{x}))$$

That is, it gives the distance to the nearest point on Γ, multiplied by the sign of $\phi(\vec{x})$. Notice that ϕ and ϕ_S will share a level set.

An example should help illustrate the idea. Consider two functions

$$\phi_1(x, y) := \sqrt{x^2 + y^2} - 1,$$

and

$$\phi_2(x, y) := x^2 + y^2 - 1.$$

Though both functions share the unit circle as their level set, ϕ_1 is a signed distance function and ϕ_2 is not. When we move radially outward from the origin we are either approaching the closest point on the level set or moving away from it. Hence, for a signed distance function ϕ with this level set, we expect that moving a unit distace radially outward will be accompanied by a unit increase in the value of ϕ. The function ϕ_1 satisfies this property, but ϕ_2 does not. The function ϕ_1 could be called the corresponding signed distance function of ϕ_2.

While signed distance functions are simple for circular level sets, for a general function ϕ the corresponding signed distance function ϕ_S is not given by an analytic formula and must be computed numerically. It is easy to visually identify a signed distance function by a plot of its contours, since successive contours will be, informally, “equally spaced” from one another. The difference is illustrated in Figure 3.2.

In our simulation we will initialize ϕ as a signed distance function. When evolving ϕ by Equation (3.1) Γ will move with the correct velocity but ϕ will no longer be a signed

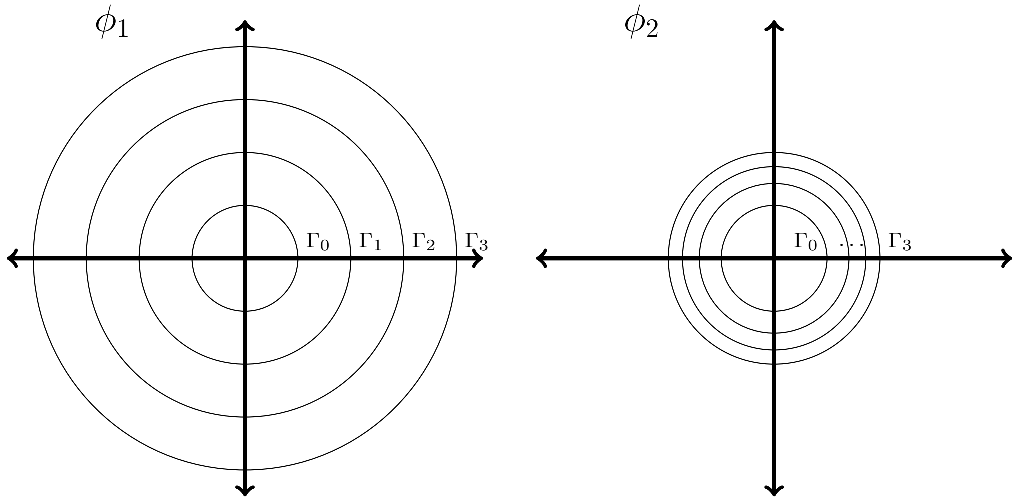

Figure 3.2: The level sets Γ_0, Γ_1, Γ_2, and Γ_3 are shown for ϕ_1 (left) and ϕ_2 (right). Notice that for ϕ_2 the concentric circular level sets are not equally spaced from one another, as is the case for ϕ_1.

distance function. In [19] the authors note that ϕ "can become irregular after some period of time". The gradients of ϕ became steep and large volume loss occurred in their water drop and air bubble simulations. They also observed incorrect qualitative behaviour of their solutions when ϕ was left to evolve only under Equation (3.1). We evolve ϕ to a signed distance function to avoid these numerical issues, since signed distance functions are well behaved and do not have extreme gradients.

The fluid simulation only depends on values of ϕ near the interface Γ. Near the interface, we would like ϕ to be well behaved, since forces such as surface tension are dependent on the curvature of Γ, which is computed using ϕ. In [19] the authors elect to evolve ϕ to its corresponding signed distance function at every time step of the fluid simulation to keep it well behaved. While evolving ϕ to a signed distance function it is important that Γ does not change since it represents the interface between the two fluids.

If we are given a level set function ϕ_0, we can evolve it to a signed distance function ϕ by solving the following problem to a steady state [6]:

$$\frac{\partial \phi}{\partial t} = S(\phi_0)\left(1 - |\nabla \phi|\right), \tag{3.3}$$

with initial condition

$$\phi(\vec{x}, 0) = \phi_0(\vec{x}), \tag{3.4}$$

where S is the sign function. The steady state solution ϕ to this PDE will share the same level set as ϕ_0. At the interface the sign function will have a sharp jump which is not amenable to finite difference discretizations. For that reason, we use a smoothed version of the sign function,

$$S_\epsilon(\phi)_{i,j} = \frac{\phi_{i,j}}{\sqrt{\phi_{i,j}^2 + \epsilon}},$$

where we use $\epsilon = \Delta x^2$ as in [6].

We can discretize Equation (3.3) in the same manner as [7, 13, 19]. We use a forward Euler time discretization to convert Equation (3.3) into the following discrete form

$$\phi_{i,j}^{n+1} = \phi_{i,j}^n - \Delta t S_\epsilon\left(\phi_{i,j}^0\right) G\left(\phi^n\right)_{i,j} \tag{3.5}$$

where

$$G(\phi)_{i,j} = \begin{cases} \sqrt{\max\left(\left(a^+\right)^2, \left(b^-\right)^2\right) + \max\left(\left(c^+\right)^2, \left(d^-\right)^2\right)} - 1, & \text{if } \phi_{i,j}^0 > 0, \\ \sqrt{\max\left(\left(a^-\right)^2, \left(b^+\right)^2\right) + \max\left(\left(c^-\right)^2, \left(d^+\right)^2\right)} - 1, & \text{if } \phi_{i,j}^0 > 0, \\ 0, & \text{otherwise.} \end{cases}$$

The "+" superscript denotes the positive part of a number and the "−" superscript denotes the negative part of a number. The values a, b, c, and d are forward and backward differences of ϕ given by,

$$\begin{aligned} a &= \frac{\phi_{i,j} - \phi_{i-1,j}}{\Delta x}, & c &= \frac{\phi_{i,j} - \phi_{i,j-1}}{\Delta y}, \\ b &= \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x}, & d &= \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y}. \end{aligned}$$

In [19], the authors noted better results when the forward and backward differences a, b, c, and d are replaced with second order ENO approximations. We use second order ENO approximations in our reinitialization as well. With the first order approximations given above, we observed curvature spikes in the reinitialized ϕ in axis-oriented directions as well as near the zero level set itself. This is particularly troublesome since we use ϕ to calculate curvature near the zero level set when implementing surface tension. With second order ENO approximations these issues were remedied. Again, we refer the reader to Appendix A for a discussion of ENO approximations.

The boundary conditions for ϕ were set as in Appendix A.6 of [7]

$$\phi_B = \phi_{B-1} + S\left(\phi_{B-1}\right)\left|\phi_{B-1} - \phi_{B-2}\right|,$$

where ϕ_B is the boundary value and ϕ_{B-1} and ϕ_{B-2} are the adjacent points in the axis direction normal to the boundary. This boundary condition keeps the characteristics of Equation (3.3) flowing outward from the level set and guarantees the sign of ϕ does not change at the boundary.

With the use of second order ENO approximations replacing a, b, c, and d this scheme is identical to the one reported by the authors of [19]. Those authors provided the following convergence criteria:

$$\frac{\sum_{|\phi_{i,j}^n|<\alpha} |\phi_{i,j}^{n+1} - \phi_{i,j}^n|}{M} < \Delta t \Delta x^2,$$

where M is the number of grid points with $|\phi_{i,j}^n| < \alpha$. That is, the number of grid points within a distance α of the level set. The authors used $\alpha = \frac{3}{2}\Delta x$ and reported convergence in typically one iteration during the fluid simulation.

In our simulations, while the level set would approximate a signed distance function in finite iterations, it did not converge under this criterion. For the purpose of the fluid simulation, where the level set is reinitialized after each time step, we elected to simply to run the reinitialization algorithm for 5 iterations. This worked well since the level set is initialized to a signed distance function and reinitialized after each time step. Hence ϕ would never deviate far from a signed distance function, and 5 iterations turned out to be sufficient for our simulations. We were careful to make sure that the reinitialization step did not change the fluid volume by checking the volume before and after reinitialization in tests of the algorithm and during early simulations.

Chapter 4

Discretization

In this chapter we will describe a *finite difference* approach to discretizing Equations (2.5) and (2.6), shown again below:

$$\frac{\partial u}{\partial t} + \frac{\partial \left(u^2\right)}{\partial x} + \frac{\partial \left(uv\right)}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} = \frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\right) + g_x,$$

$$\frac{\partial v}{\partial t} + \frac{\partial \left(uv\right)}{\partial x} + \frac{\partial \left(v^2\right)}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} = \frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\right) + g_y.$$

Discretization refers to moving from a continuous problem to one considered only at finitely many points [8]. When we discretize time and space we need a way to approximate the terms found in the equations of motion. Unless otherwise indicated, we will largely follow along with the discretizations found in [8].

4.1 Discretization of the Navier-Stokes Equations

We will be considering a rectangular computational domain Ω. For finite difference methods, we typically discretize a rectangular region into a rectilinear grid, with grid sizes Δx and Δy in the x and y directions respectively. We will focus on the square cells defined by the grid lines, associating the indices i and j with cells $c_{i,j}$, for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$. We illustrate this in the left panel of Figure 4.1.

When discretizing the Navier-Stokes equations we employ what is known as a *staggered grid*, as can be seen in [3, 4, 8, 17, 19, 20]. The main idea of the staggered grid is not to

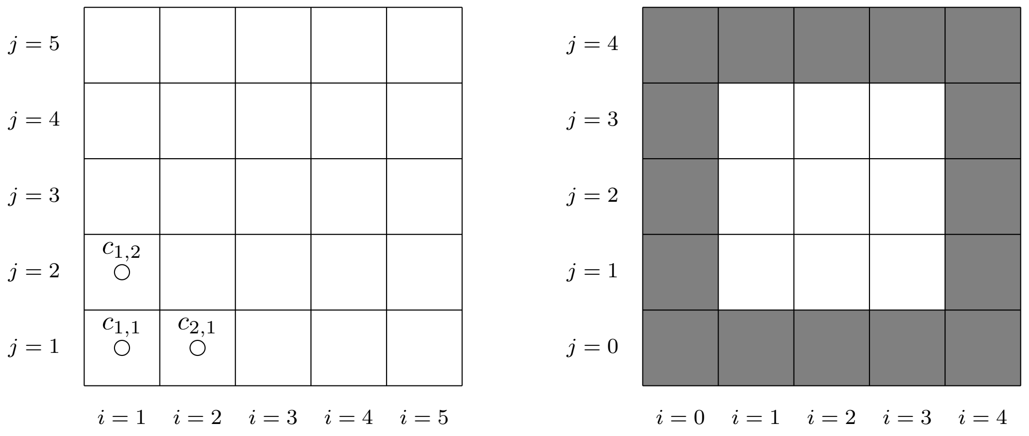

Figure 4.1: On the left, rectilinear grid lines divide the rectangular domain into cells $c_{i,j}$ for $i = 1, \ldots, n_x$, $j = 1, \ldots, n_y$, and $n_x = n_y = 5$. On the right we have a rectangular domain surrounded by ghost cells with $n_x = n_y = 3$, in which the extremal indices $i = 0$, $i = n_x + 1$, $j = 0$, and $j = n_y + 1$ correspond to the ghost cells.

store the parameters for cell $c_{i,j}$ at the same position. We store pressure $p_{i,j}$ at the center of cell $c_{i,j}$. The x-velocity $u_{i,j}$ is stored at the midpoint of the right edge of cell $c_{i,j}$. Finally, the y-velocity $v_{i,j}$ is stored at the midpoint of the top edge of cell $c_{i,j}$. This is shown in Figure 4.2.

The purpose of the staggered grid set up is to prevent possible pressure oscillations which could occur were the unknowns evaluated at the same grid points. A consequence of the staggered grid is that extremal values of the parameters do not all lie on the boundaries of the physical domain. For instance, there are no v values on the left or right boundaries. Similarly, there are no u values on the top or bottom boundaries. Since p values are stored at cell centers, none are stored on any of the boundaries.

Since we will need to specify boundary conditions for the simulation, we need a way of defining u, v, and p at the boundary. This problem is resolved by surrounding the physical domain by a layer of *ghost cells*. These cells store parameters u, v, and p in the same manner as the original *physical cells*. Whenever a parameter value is needed on the boundary, but is not explicitly defined there, it is found by averaging the parameter value in the adjacent physical and ghost cells. We can refer to ghost cells using the same indexing as for the physical cells, except that the ghost cells correspond the extremal values $i = 0$, $i = n_x + 1$, $j = 0$, and $j = n_y + 1$. A diagram of a domain with ghost cells can be seen in the right panel of Figure 4.1.

Now we can discuss the discretization of the Navier-Stokes equations. The continuity

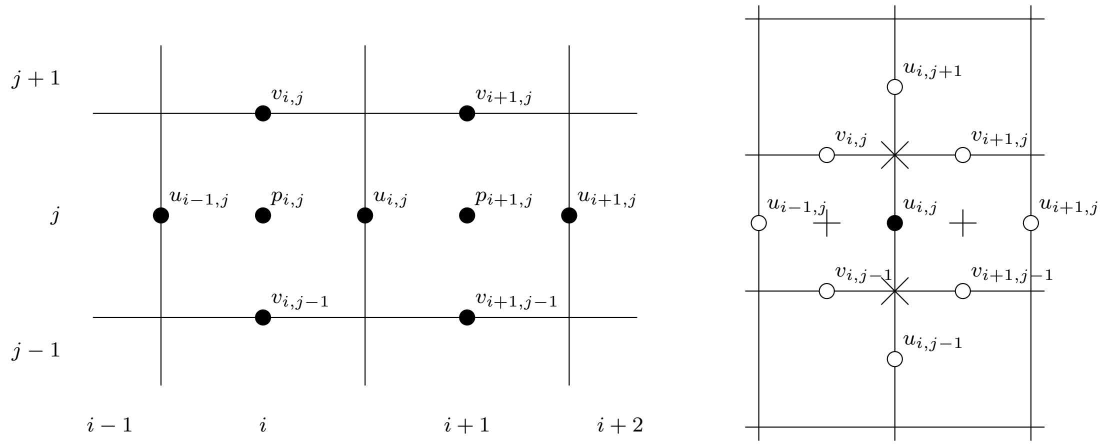

Figure 4.2: The left panel shows a staggered grid setup with variables p, u, and v stored at different positions within each cell. The right panel shows the velocity values needed to update $u_{i,j}$ using the momentum equation for u, Equation (2.5).

Equation (2.3) is discretized at the center of cells $c_{i,j}$ for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$. We need approximations to derivatives $\partial u/\partial x$ and $\partial v/\partial y$. We use centered differences given by,

$$\left[\frac{\partial u}{\partial x}\right]_{i,j} := \frac{u_{i,j} - u_{i-1,j}}{\Delta x}, \quad \text{and} \quad \left[\frac{\partial v}{\partial y}\right]_{i,j} := \frac{v_{i,j} - v_{i,j-1}}{\Delta y}. \tag{4.1}$$

These look like backward differences, but recall that $u_{i,j}$ and $v_{i,j}$ are located at the midpoints of the right and top cell edge respectively, not at the cell center.

The momentum Equation (2.5) for u is discretized at the midpoint of the right cell edges. Similarly, the momentum Equation (2.6) for v is discretized at the midpoint of the top cell edges.

The terms $\dfrac{\partial}{\partial x}\left(\mu\dfrac{\partial u}{\partial x}\right)$, $\dfrac{\partial}{\partial y}\left(\mu\dfrac{\partial u}{\partial y}\right)$, $\dfrac{\partial}{\partial x}\left(\mu\dfrac{\partial v}{\partial x}\right)$, and $\dfrac{\partial}{\partial y}\left(\mu\dfrac{\partial v}{\partial y}\right)$, known as the *diffusive* terms, show up in the momentum Equations (2.5) and (2.6), as a result of the stress tensor $\boldsymbol{\sigma}$. The discretization of these terms needs to be handled carefully since μ changes at the interface between the two fluids. The level set function ϕ will be used to make the discretizations. We will discuss how to handle the diffusive terms after discussing how to solve for the pressure, where similar ideas are used. The finite difference formulas for these terms will be covered in Section 4.2.3.

The terms $\partial\left(u^2\right)/\partial x$, $\partial\left(uv\right)/\partial y$, $\partial\left(uv\right)/\partial x$, and $\partial\left(v^2\right)/\partial y$ form the *convective* terms. The difference formulas for the convective terms are somewhat complicated, so we will handle those which appear in Equation (2.5) first, and then move on to Equation (2.6).

Equation (2.5) is the momentum equation for u. Again, we discretize Equation (2.5) at the midpoint of the right cell edges. The derivatives $\partial\left(u^2\right)/\partial x$ and $\partial\left(uv\right)/\partial y$ show up in Equation (2.5) and will require the velocity values shown in the left panel of Figure 4.2.

To discretize $\partial(uv)/\partial y$, we will need appropriate values of the product uv above and below the position of $u_{i,j}$. One solution is to compute the product uv at the points marked with an $\times$ using the average values of u and v, leading to the discretization:

$$\left[\frac{\partial(uv)}{\partial y}\right]_{i,j} := \frac{1}{\Delta y}\left(\frac{(v_{i,j}+v_{i+1,j})}{2}\frac{(u_{i,j}+u_{i,j+1})}{2} - \frac{(v_{i,j-1}+v_{i+1,j-1})}{2}\frac{(u_{i,j-1}+u_{i,j})}{2}\right). \tag{4.2}$$

In a similar manner, we can use central differencing to discretize the $\partial(u^2)/\partial x$ term using averaged values of u at the positions marked by a $+$ in the right panel of Figure 4.2. This leads to the following discretization:

$$\left[\frac{\partial(u^2)}{\partial x}\right]_{i,j} := \frac{1}{\Delta x}\left(\left(\frac{u_{i,j}+u_{i+1,j}}{2}\right)^2 - \left(\frac{u_{i-1,j}+u_{i,j}}{2}\right)^2\right). \tag{4.3}$$

For stability purposes, the convective terms need to be approximated by a mixture of the centered differences given in Equations (4.2) and (4.3) and *donor-cell* discretizations. The resulting formulas are as follows:

$$\begin{aligned}\left[\frac{\partial(u^2)}{\partial x}\right]_{i,j} &:= \frac{1}{\Delta x}\left(\left(\frac{u_{i,j}+u_{i+1,j}}{2}\right)^2 - \left(\frac{u_{i-1,j}+u_{i,j}}{2}\right)^2\right)\\ &+ \gamma\frac{1}{\Delta x}\left(\frac{|u_{i,j}+u_{i+1,j}|}{2}\frac{(u_{i,j}-u_{i+1,j})}{2} - \frac{|u_{i-1,j}+u_{i,j}|}{2}\frac{(u_{i-1,j}-u_{i,j})}{2}\right)\end{aligned} \tag{4.4}$$

$$\begin{aligned}\left[\frac{\partial(uv)}{\partial y}\right]_{i,j} &:= \frac{1}{\Delta y}\left(\frac{(v_{i,j}+v_{i+1,j})}{2}\frac{(u_{i,j}+u_{i,j+1})}{2} - \frac{(v_{i,j-1}+v_{i+1,j-1})}{2}\frac{(u_{i,j-1}+u_{i,j})}{2}\right)\\ &+ \gamma\frac{1}{\Delta y}\left(\frac{|v_{i,j}+v_{i+1,j}|}{2}\frac{(u_{i,j}-u_{i,j+1})}{2} - \frac{|v_{i,j-1}+v_{i+1,j-1}|}{2}\frac{(u_{i,j-1}-u_{i,j})}{2}\right)\end{aligned} \tag{4.5}$$

The convective terms for Equation (2.6) are treated in a symmetric manner. For the sake of clarity and completeness, these formulas are given below:

$$\begin{aligned}\left[\frac{\partial(uv)}{\partial x}\right]_{i,j} &:= \frac{1}{\Delta x}\left(\frac{(u_{i,j}+u_{i,j+1})}{2}\frac{(v_{i,j}+v_{i+1,j})}{2} - \frac{(u_{i-1,j}+u_{i-1,j+1})}{2}\frac{(v_{i-1,j}+v_{i,j})}{2}\right)\\ &+ \gamma\frac{1}{\Delta x}\left(\frac{|u_{i,j}+u_{i,j+1}|}{2}\frac{(v_{i,j}-v_{i+1,j})}{2} - \frac{|u_{i-1,j}+u_{i-1,j+1}|}{2}\frac{(v_{i-1,j}-v_{i,j})}{2}\right)\end{aligned} \tag{4.6}$$

$$\begin{aligned}\left[\frac{\partial(v^2)}{\partial y}\right]_{i,j} &:= \frac{1}{\Delta y}\left(\left(\frac{v_{i,j}+v_{i,j+1}}{2}\right)^2 - \left(\frac{v_{i,j-1}+v_{i,j}}{2}\right)^2\right)\\ &+ \gamma\frac{1}{\Delta y}\left(\frac{|v_{i,j}+v_{i,j+1}|}{2}\frac{(v_{i,j}-v_{i,j+1})}{2} - \frac{|v_{i,j-1}+v_{i,j}|}{2}\frac{(v_{i,j-1}-v_{i,j})}{2}\right)\end{aligned} \tag{4.7}$$

The parameter γ in Equations (4.4), (4.5), (4.6) and (4.7) is chosen between 0 and 1 and represents the balance between the centered difference scheme and donor-cell discretization scheme. When $\gamma = 0$ the formulas are purely centered differences, and when $\gamma = 1$ the formulas are purely donor-cell discretizations. In our simulations we used $\gamma = 0.9$. In general, γ should be chosen so that,

$$\gamma \geq \max_{i,j}\left(\frac{|u_{i,j}\Delta t|}{\Delta x}, \frac{|v_{i,j}\Delta t|}{\Delta y}\right).$$

We also need spatial pressure derivatives. Equation (2.5) requires a discretization of $\partial p/\partial x$ and Equation (2.6) requires a discretization of $\partial p/\partial y$. We use the centered differences

$$\left[\frac{\partial p}{\partial x}\right]_{i,j} := \frac{p_{i+1,j}-p_{i,j}}{\Delta x}, \quad \text{and} \quad \left[\frac{\partial p}{\partial y}\right]_{i,j} := \frac{p_{i,j+1}-p_{i,j}}{\Delta y}. \tag{4.8}$$

Note that these are centered differences because we discretize Equation (2.5) at the midpoints of right cell edges (like the u values), and we discretize Equation (2.6) at the midpoints of top cell edges (like the v values).

4.1.1 Boundary Values

It was mentioned in Section 4.1 that we will require specifications of the values,

$$\begin{array}{lll} u_{0,j}, & u_{n_x,j}, & j = 1,\ldots,n_y,\\ v_{i,0}, & v_{i,n_y}, & i = 1,\ldots,n_x,\end{array}$$

on the boundary and,

$$\begin{aligned} &u_{i,0}, \quad u_{i,n_y+1}, \quad i = 1, \ldots, n_x, \\ &v_{0,j}, \quad v_{n_x+1,j}, \quad j = 1, \ldots, n_y, \end{aligned}$$

outside the physical domain Ω. We will consider how to impose two different types of solid wall boundary conditions below.

1. *No-slip condition:* In the no-slip condition, velocity should be zero at the boundary. This is to ensure fluid never leaves the domain Ω and to ensure the fluid sticks (or, does not slip) along the boundary. The values which are directly on the boundary already are thus set to zero.

$$\begin{aligned} u_{0,j} = 0, \quad u_{n_x,j} = 0, \quad j = 1, \ldots, n_y, \\ v_{i,0} = 0, \quad v_{i,n_y} = 0, \quad i = 1, \ldots, n_x. \end{aligned} \tag{4.9}$$

   Considering the vertical boundaries, we have no v values and must average them from the ghost and physical cells. Similarly, on the horizontal boundaries, we must average ghost and physical cell values of u to get boundary values. In both cases we average the two values and want the result to equal zero. This leads to the following conditions:

$$\begin{aligned} u_{i,0} = -u_{i,1}, \quad u_{i,n_y+1} = -u_{i,n_y}, \quad i = 1, \ldots, n_x, \\ v_{0,j} = -v_{1,j}, \quad v_{n_x+1,j} = -v_{n_x,j}, \quad j = 1, \ldots, n_y. \end{aligned} \tag{4.10}$$

2. *Free-slip condition:* In the free slip condition, velocity normal to the boundary should be zero, as well as the normal derivative of the velocity at the boundary. This corresponds to a solid wall which allows fluid to flow across it without resistance. The velocity values lying directly on a boundary are also those normal to the boundary. For instance, the y-velocities v lie on the horizontal boundaries. Hence we get the same conditions for the values directly on the boundary as in the no-slip condition.

$$\begin{aligned} u_{0,j} = 0, \quad u_{n_x,j} = 0, \quad j = 1, \ldots, n_y, \\ v_{i,0} = 0, \quad v_{i,n_y} = 0, \quad i = 1, \ldots, n_x. \end{aligned} \tag{4.11}$$

   To impose that the normal derivative at the boundary be zero, we take a centered difference of the velocity centered at the boundary and set it equal to zero. This gives the following conditions:

$$\begin{aligned} u_{i,0} = u_{i,1}, \quad u_{i,n_y+1} = u_{i,n_y}, \quad i = 1, \ldots, n_x, \\ v_{0,j} = v_{1,j}, \quad v_{n_x+1,j} = v_{n_x,j}, \quad j = 1, \ldots, n_y. \end{aligned} \tag{4.12}$$

4.1.2 Time Discretization

To simulate the fluid flow for times $t \in [0, T]$ we begin our simulation with initial conditions for all parameters (p, u, and v) at time $t_0 = 0$. The parameters p, u, and v are calculated at the next time step $t_1 = t_0 + \Delta t$ using spatial derivatives from the previous time step t_0. This process is repeated until we reach a time step $t_n = t_{n-1} + \Delta t = T$. Such a method would be an *explicit* method since the parameter values at time t_{n+1} can be found using only the parameter values from time t_n. In contrast, an *implicit* method would use spatial derivatives from both time step t_{n+1} and t_n. We will be using an explicit method in our simulations.

The time derivatives from Equations (2.5) and (2.6) are discretized using *Euler's method*. Euler's method uses first order differences,

$$\left[\frac{\partial u}{\partial t}\right]^{n+1} := \frac{u^{n+1} - u^n}{\Delta t}, \qquad \left[\frac{\partial v}{\partial t}\right]^{n+1} := \frac{v^{n+1} - v^n}{\Delta t},$$

where the superscripts denote the time level.

The simulation will become unstable if Δt is too large. We need the following three conditions:

$$\frac{2\Delta t}{Re} < \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)^{-1}, \quad |u_{max}|\Delta t < \Delta x, \quad |v_{max}|\Delta t < \Delta y. \tag{4.13}$$

The $|u_{max}|$ and $|v_{max}|$ terms are the maximum absolute velocities in the x and y directions respectively. The last two restrictions correspond to the typical *Courant-Friedrichs-Lewy*(CFL) conditions. The first condition uses the Reynolds number, Re. Our simulations deal with a liquid and air region. Let ρ_A and μ_A denote the density and viscosity of the air, and let ρ_L and μ_L denote the density and viscosity of the liquid. Then the Reynolds number for the time step restriction is given by,

$$Re = \min\left\{\frac{\rho_A u_\infty L}{\mu_A}, \frac{\rho_L u_\infty L}{\mu_L}\right\}. \tag{4.14}$$

When surface tension is incorporated in our simulation we have to impose [19],

$$\Delta t < \Delta t_s := \sqrt{(\rho_A + \rho_L)/8\pi\sigma} \cdot \min\left\{\Delta x, \Delta y\right\}^{\frac{3}{2}},$$

where σ is a constant coefficient which determines the strength of surface tension. Surface tension will be discussed in Section 4.2.2 in the context of the *variable coefficient Poisson equation* for pressure, Equation (4.18).

The restrictions above can be summarized as,

$$\Delta t := \tau \min \left\{ \frac{Re}{2} \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right)^{-1}, \frac{\Delta x}{|u_{max}|}, \frac{\Delta y}{|v_{max}|}, \Delta t_s \right\}, \tag{4.15}$$

where $\tau \in (0, 1)$ is a safety factor ensuring we do not get too close to the upper limits. In our simulations we found that $\tau = 0.5$ effectively enforced stability.

4.1.3 Discretization and Initialization of Level Set Function ϕ

The level set function ϕ is discretized at the centers of the cells $c_{i,j}$, just like the pressure p. When initializing ϕ, we need to do so in a way that splits the domain Ω into two fluid regions. For our simulations, we used the convention that $\phi \leq 0$ corresponds to a liquid region and $\phi > 0$ corresponds to an air region. The exact location of the boundary depends on how ϕ is initialized. Our work was focused on drop simulations where a circular drop of liquid falls into a pool of fluid below. Hence, we initialized ϕ so that its level set was a circle positioned over a horizontal line. The origin is located at the bottom left corner of the simulation domain. If we want the circle to have radius R and the pool to be at a height of h then we can initialize ϕ as

$$\phi(x, y) = \min \left\{ y - h, \sqrt{(x - x_c)^2 + (y - y_c)^2} - R \right\},$$

where (x_c, y_c) is the center of the drop. This gives a well behaved ϕ which is similar to a signed distance function.

Note that since ϕ values are stored at cell centres, when we advect ϕ we must linearly interpolate velocities u and v at the cell centres. This means that the upwinding formulas from Section 3.3 become

$$(\phi_x)_{i,j} = \begin{cases} \dfrac{\phi^n_{i,j} - \phi^n_{i-1,j}}{\Delta x}, & \text{if } \dfrac{u_{i,j} + u_{i-1,j}}{2} > 0, \\ \dfrac{\phi^n_{i+1,j} - \phi^n_{i,j}}{\Delta x}, & \text{if } \dfrac{u_{i,j} + u_{i-1,j}}{2} \leq 0, \end{cases}$$

and

$$(\phi_y)_{i,j} = \begin{cases} \dfrac{\phi^n_{i,j} - \phi^n_{i,j-1}}{\Delta y}, & \text{if } \dfrac{v_{i,j} + v_{i,j-1}}{2} > 0, \\ \dfrac{\phi^n_{i,j+1} - \phi^n_{i,j}}{\Delta y}, & \text{if } \dfrac{v_{i,j} + v_{i,j-1}}{2} \leq 0. \end{cases}$$

Similarly, the values of u and v in Equation (3.2) use cell centered velocities $\dfrac{u_{i,j}+u_{i-1,j}}{2}$ and $\dfrac{v_{i,j}+v_{i,j-1}}{2}$ for cell $c_{i,j}$.

4.2 Algorithm Description

We have most of the tools needed to implement the proposed fluid simulator algorithm. The only missing pieces are the spatial derivatives arising from the stress tensor, whose discretizations are similar to those used when solving for pressure. We will first discuss the algorithm so that we can see how pressure is computed.

4.2.1 The Time Stepping Outline

As noted in Section 4.1.2, we first initialize the values of p, u, and v at time $t = 0$. In our simulations, we initialize these parameters uniformly to zero. We also want to initialize ϕ as described in Section 4.1.3.

We first perform the time discretization in the momentum Equations (2.5) and (2.6). This gives the two equations,

$$\frac{u^{n+1}-u^n}{\Delta t}+\frac{\partial\left(u^2\right)}{\partial x}+\frac{\partial\left(uv\right)}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial x}=\frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\right)+g_x,$$

and

$$\frac{v^{n+1}-v^n}{\Delta t}+\frac{\partial\left(uv\right)}{\partial x}+\frac{\partial\left(v^2\right)}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial y}=\frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\right)+g_y.$$

We isolate the u^{n+1} and v^{n+1} terms in the equations above to get,

$$u^{n+1}=u^n+\Delta t\left(\frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\right)-\frac{\partial\left(u^2\right)}{\partial x}-\frac{\partial\left(uv\right)}{\partial y}+g_x-\frac{1}{\rho}\frac{\partial p}{\partial x}\right),$$

and

$$v^{n+1}=v^n+\Delta t\left(\frac{1}{\rho u_\infty L}\left(\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)\right)-\frac{\partial\left(uv\right)}{\partial x}-\frac{\partial\left(v^2\right)}{\partial y}+g_y-\frac{1}{\rho}\frac{\partial p}{\partial y}\right).$$

Define the following variables:

$$\begin{aligned}
F &:= u^n + \Delta t \left(\frac{1}{\rho u_\infty L} \left(\frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \right) - \frac{\partial \left(u^2\right)}{\partial x} - \frac{\partial \left(uv\right)}{\partial y} + g_x \right), \\
G &:= v^n + \Delta t \left(\frac{1}{\rho u_\infty L} \left(\frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) \right) - \frac{\partial \left(uv\right)}{\partial x} - \frac{\partial \left(v^2\right)}{\partial y} + g_y \right).
\end{aligned} \tag{4.16}$$

With these variables, the equations we are working on become,

$$\begin{aligned}
u^{n+1} &= F - \frac{\Delta t}{\rho} \frac{\partial p}{\partial x}, \\
v^{n+1} &= G - \frac{\Delta t}{\rho} \frac{\partial p}{\partial y}.
\end{aligned} \tag{4.17}$$

To fully discretize Equation (4.17), we associate F and G with time level t_n, and the pressure derivatives with time step t_{n+1}. In this way our discretization can be considered a semi-implicit scheme, which is explicit in velocities and implicit in pressure. We will thus need a way of computing the pressure at time level t_{n+1}.

The pressure is found by making use of the continuity equation $\nabla \cdot \vec{u} = 0$. We can apply the continuity equation at time t_{n+1} to get

$$0 = \frac{\partial u^{n+1}}{\partial x} + \frac{\partial v^{n+1}}{\partial y} = \frac{\partial F^n}{\partial x} - \Delta t \frac{\partial}{\partial x} \left(\frac{1}{\rho} \frac{\partial p^{n+1}}{\partial x} \right) + \frac{\partial G^n}{\partial y} - \Delta t \frac{\partial}{\partial y} \left(\frac{1}{\rho} \frac{\partial p^{n+1}}{\partial y} \right).$$

We can rearrange this to get the *variable coefficient Poisson equation* for pressure at time t_{n+1}:

$$\frac{\partial}{\partial x} \left(\frac{1}{\rho} \frac{\partial p^{n+1}}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\rho} \frac{\partial p^{n+1}}{\partial y} \right) = \frac{1}{\Delta t} \left(\frac{\partial F^n}{\partial x} + \frac{\partial G^n}{\partial y} \right) \tag{4.18}$$

In vector form this could be written as

$$\nabla \cdot \left(\frac{1}{\rho} \nabla p^{n+1} \right) = \frac{1}{\Delta t} \nabla \cdot H^n, \tag{4.19}$$

where $H^n = \begin{pmatrix} F^n \\ G^n \end{pmatrix}$.

In summary, to get from time step t_n to time step t_{n+1} we follow these steps:

1. Compute F^n and G^n according to Equation (4.16) using velocities u^n and v^n.

2. Solve the *variable coefficient Poisson equation* (4.18) for pressure p^{n+1}.
3. Compute the new velocities u^{n+1} and v^{n+1} using (4.17).

We still need to discuss steps one and two a bit more carefully. For step one, we still need to specify boundary conditions for F and G. We also still need to see the difference formulas for the diffusive terms showing up in F and G. For step two, we still need to describe how to solve for the pressure, which involves solving a linear system of equations. The discussion of how to solve for the pressure will give us the tools needed to discretize the diffusive terms.

4.2.2 Solving the Variable Coefficient Poisson Equation for Pressure

We update pressure from time t_n to t_{n+1} by solving the variable coefficient Poisson equation (4.18). We will begin by considering the one dimensional case which will naturally extend to two dimensions. In one dimension, the variable coefficient pressure equation becomes

$$\frac{d}{dx}\left(\frac{1}{\rho}\frac{dp}{dx}\right) = \frac{1}{\Delta t}\frac{dF}{dx}. \tag{4.20}$$

The question is how to form a discretization of $\frac{d}{dx}\left(\frac{1}{\rho}\frac{dp}{dx}\right)$ at a point x_k that captures the jump at the interface Γ in density, pressure gradient, and, in the case of surface tension, pressure. These interface conditions are illustrated in the left panel of Figure 4.3, where p_I is the pressure at the interface, κ is the curvature at the interface, and σ is a constant controlling the strength of the surface tension. Curvature is calculated at cell centers (the same locations at which ϕ is stored) using the formula $\kappa = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|}\right)$, and standard centered differences.

Consider Figure 4.3. Suppose that x_k belongs to the liquid region with density ρ_L, and x_{k+1} belongs to the air region with density ρ_A. The exact interface between liquid and air is where $\phi = 0$ at position $x_k + \theta\Delta x$. We can approximate the location of the interface using a linear interpolation of $\phi_k = \phi(x_k)$ and $\phi_{k+1} = \phi(x_{k+1})$. This gives,

$$\theta = \frac{\phi_k}{\phi_k - \phi_{k+1}}. \tag{4.21}$$

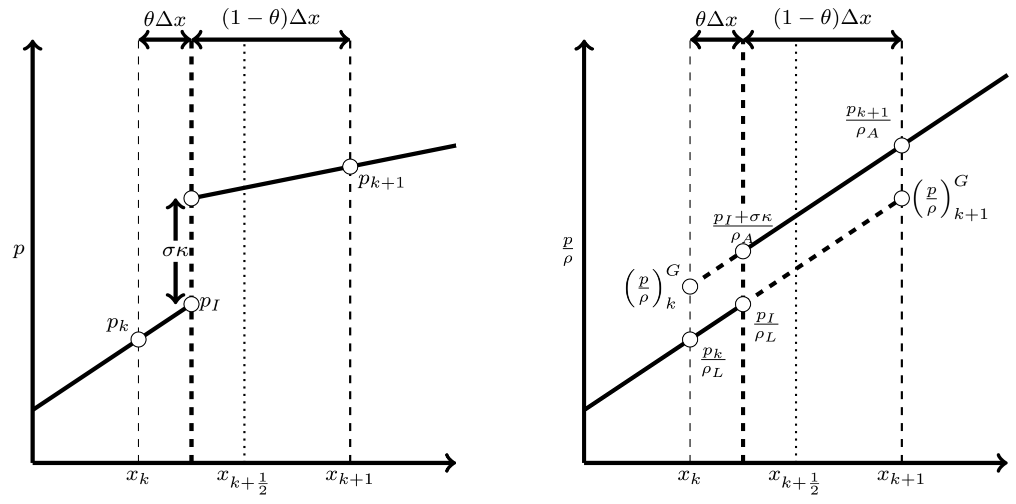

Figure 4.3: The left panel shows a plot of pressure across a fluid interface. The jump in pressure and pressure gradient can be seen at the interface between x_k and $x_{k+\frac{1}{2}}$ (left). The jump in pressure is a result of surface tension, and is proportional to the curvature κ at the interface. When we look at the plot of p/ρ (right) our discretization assumes no jump in the gradient of p/ρ, but the jump in the value of p/ρ still exists. We use linear extensions of the pressure to find the "ghost" values of pressure, denoted with the G superscript.

We will use a centered discretization scheme for $\dfrac{d}{dx}\left(\dfrac{1}{\rho}\dfrac{dp}{dx}\right)$ at x_k, which takes the form

$$\left[\frac{d}{dx}\left(\frac{1}{\rho}\frac{dp}{dx}\right)\right]_k = \frac{\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}} - \left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k-\frac{1}{2}}}{\Delta x}. \tag{4.22}$$

The terms $\left[\dfrac{1}{\rho}\dfrac{dp}{dx}\right]_{k+\frac{1}{2}}$ and $\left[\dfrac{1}{\rho}\dfrac{dp}{dx}\right]_{k-\frac{1}{2}}$ are centered differences of $\dfrac{1}{\rho}\dfrac{dp}{dx}$ with mesh width $\Delta x/2$. It will help to make these approximations if we look at the graph of p/ρ, shown in the right panel of Figure 4.3. Our assumptions and description of the discretization will follow [3] closely. We assume no jump in the slope of p/ρ, and a jump of size $\sigma\kappa/\rho_A$.

In our example, x_k and x_{k-1} both lie in the liquid domain and so the density is ρ_L at these locations. We can hence discretize $\left[\dfrac{1}{\rho}\dfrac{dp}{dx}\right]_{k-\frac{1}{2}}$ as,

$$\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k-\frac{1}{2}} = \frac{1}{\rho_L}\frac{p_k - p_{k-1}}{\Delta x}. \tag{4.23}$$

For the term $\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}}$, we need to be more careful. Note that if we knew the value of p_I we could define our approximation as,

$$\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}} = \frac{1}{\rho_L}\frac{p_I - p_k}{\theta \Delta x}. \tag{4.24}$$

To find p_I, notice that by comparing the slopes of the solid black lines in the left panel of Figure 4.3 we get

$$\frac{1}{\rho_L}\frac{p_I - p_k}{\theta \Delta x} = \frac{1}{\rho_A}\frac{p_{k+1} - (p_I + \sigma\kappa)}{(1-\theta)\Delta x}.$$

We can rearrange this to get,

$$p_I = \frac{\rho_L (p_{k+1} - \sigma\kappa)\,\theta + \rho_A p_k (1-\theta)}{\rho_L\theta + \rho_A (1-\theta)}.$$

which can be substituted back into Equation (4.24) to get,

$$\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}} = \frac{1}{\hat{\rho}}\frac{p_{k+1} - p_k - \sigma\kappa}{\Delta x}, \tag{4.25}$$

where

$$\hat{\rho} = \rho_L\theta + \rho_A (1-\theta). \tag{4.26}$$

We can substitute Equations (4.23) and (4.25) into Equation (4.22) to get,

$$\left[\frac{d}{dx}\left(\frac{1}{\rho}\frac{dp}{dx}\right)\right]_k = \frac{\frac{1}{\hat{\rho}}\frac{p_{k+1}-p_k-\sigma\kappa}{\Delta x} - \frac{1}{\rho_L}\frac{p_k - p_{k-1}}{\Delta x}}{\Delta x}. \tag{4.27}$$

We can substitute the left hand side of Equation (4.20) with the right hand side of Equation (4.27), and move the surface tension to the right hand side to get,

$$\frac{1}{\hat{\rho}}\frac{p_{k+1} - p_k}{\Delta x^2} - \frac{1}{\rho_L}\frac{p_k - p_{k-1}}{\Delta x^2} = \frac{1}{\Delta t}\frac{dF}{dx} + \frac{1}{\hat{\rho}}\frac{\sigma\kappa}{\Delta x^2}. \tag{4.28}$$

Similarly, if the interface were located between x_k and x_{k-1}, we would have,

$$\frac{1}{\rho_L}\frac{p_{k+1} - p_k}{\Delta x^2} - \frac{1}{\hat{\rho}}\frac{p_k - p_{k-1}}{\Delta x^2} = \frac{1}{\Delta t}\frac{dF}{dx} + \frac{1}{\hat{\rho}}\frac{\sigma\kappa}{\Delta x^2}, \tag{4.29}$$

where $\hat{\rho}$ is calculated in a similar manner as above.

Note that the terms $\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}}$ and $\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k-\frac{1}{2}}$ are discretized independently of each other. The term $\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k+\frac{1}{2}}$ is a standard centered discretization about $x_{k+\frac{1}{2}}$ unless the interface lies between x_k and x_{k+1}, in which case the density ρ becomes a weighted average between ρ_L and ρ_A. Similarly, the $\left[\frac{1}{\rho}\frac{dp}{dx}\right]_{k-\frac{1}{2}}$ term is a standard centered discretization about $x_{k-\frac{1}{2}}$ unless the interface lies between x_k and x_{k-1}, in which case the density ρ becomes a weighted average between ρ_L and ρ_A. Hence we can consider the left and right arms of the finite difference stencil for $\left[\frac{d}{dx}\left(\frac{1}{\rho}\frac{dp}{dx}\right)\right]_k$ separately. We can also keep all surface tension terms on the right hand side of Equation (4.20).

In two dimensions, we can still discretize the $\left[\frac{\partial}{\partial x}\left(\frac{1}{\rho}\frac{\partial p}{\partial x}\right)\right]_{i,j}$ term in the same manner as described above. The k index from the discussion above now corresponds to the i index. The term $\left[\frac{\partial}{\partial y}\left(\frac{1}{\rho}\frac{\partial p}{\partial y}\right)\right]_{i,j}$ can also be discretized just as above, but now the derivatives are in the y-direction, and hence the k index from above corresponds to the j index.

Readers should also note that we discussed the case where x_k (or in two dimensions $x_{i,j}$) lied in the liquid domain. In the case that x_k lies in the air region, the discretization turns out to be identical except with the roles of ρ_L and ρ_A switched. However, the surface tension terms reverse sign when x_k (or $x_{i,j}$) lies in the air region. This is because the jump in pressure from liquid to air becomes a drop in pressure from air to liquid. For a thorough discussion of the variable coefficient Poisson equation

$$\nabla \cdot (\beta(\vec{x})\nabla p(\vec{x})) = f(\vec{x})$$

on irregular domains Ω defined with level sets, the reader is referred to [11]. The discussion is more general, but when adapted for our purposes, gives the same discretization formulas.

We now have a discretization of the left hand side of Equation (4.18) for all points $x_{i,j}$, $i = 1, \ldots, n_x$, $j = 1, \ldots, n_y$. The right hand side is more simply discretized as,

$$\frac{1}{\Delta t}\left(\frac{\partial F^n}{\partial x} + \frac{\partial G^n}{\partial y}\right) = \frac{1}{\Delta t}\left(\frac{F^n_{i,j} - F^n_{i-1,j}}{\Delta x} + \frac{G^n_{i,j} - G^n_{i,j-1}}{\Delta y}\right), \tag{4.30}$$

for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$. These discretizations form a linear system of equations when the surface tension terms are moved to the right hand side. To be well defined, we need boundary values of p, F, and G. For the staggered grid setup, we use the same boundary conditions as in [8], which come from multiplying Equations (4.33) and (4.34) (introduced in the upcoming section) with the unit exterior normal vector $\vec{n}$ on the boundary of the physical domain. The details are omitted, and the resulting boundary conditions are,

$$\begin{aligned} p_{0,j} = p_{1,j}, \quad p_{n_x+1,j} = p_{n_x,j}, \quad j = 1, \ldots, n_y, \\ p_{i,0} = p_{i,1}, \quad p_{i,n_y+1} = p_{i,n_y}, \quad i = 1, \ldots, n_x, \end{aligned} \tag{4.31}$$

and

$$\begin{aligned} F_{0,j} = u_{0,j}, \quad F_{n_x,j} = u_{n_x,j}, \quad j = 1, \ldots, n_y, \\ G_{i,0} = v_{i,0}, \quad G_{i,n_y} = v_{i,n_y}, \quad i = 1, \ldots, n_x, \end{aligned} \tag{4.32}$$

for p, F, and G.

The linear equations above form a symmetric "negative definite" matrix system. Taking the negative of the corresponding matrix equation gives a symmetric positive definite system which can be solved with the iterative methods Conjugate Gradient (CG), Preconditioned Conjugate Gradient (PCG), or Multigrid (MG) [14, 19].

4.2.3 Fully Discretized Momentum Equations

Using the pressure derivatives from Equation (4.8) and the form of the momentum equations shown in Equation (4.17), we can get

$$u_{i,j}^{n+1} = F_{i,j}^{n} - \frac{\Delta t}{\rho \Delta x} \left(p_{i+1,j}^{n+1} - p_{i,j}^{n+1} \right), \tag{4.33}$$

for $i = 1, \ldots, n_x - 1$, and $j = 1, \ldots, n_y$, as well as

$$v_{i,j}^{n+1} = G_{i,j}^{n} - \frac{\Delta t}{\rho \Delta y} \left(p_{i,j+1}^{n+1} - p_{i,j}^{n+1} \right), \tag{4.34}$$

for $i = 1, \ldots, n_x$, and $j = 1, \ldots, n_y - 1$. As noted previously, F is discretized at the midpoint of right cell edges, just as the u values are. G is discretized at the midpoint of the top cell edges, just as the v values are. The discrete versions of F and G are then

$$\begin{aligned} F_{i,j} \;&:=\; u_{i,j} \\ &+\; \Delta t \left(\frac{1}{\rho u_\infty L} \left(\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) \right]_{i,j} + \left[\frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \right]_{i,j} \right) - \left[\frac{\partial (u^2)}{\partial x} \right]_{i,j} - \left[\frac{\partial (uv)}{\partial y} \right]_{i,j} + g_x \right), \end{aligned} \tag{4.35}$$

and

$$\begin{aligned} G_{i,j} &:= v_{i,j} \\ &+ \Delta t \left(\frac{1}{\rho u_\infty L} \left(\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) \right]_{i,j} + \left[\frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) \right]_{i,j} \right) - \left[\frac{\partial (uv)}{\partial x} \right]_{i,j} - \left[\frac{\partial (v^2)}{\partial y} \right]_{i,j} + g_y \right). \end{aligned} \tag{4.36}$$

We defined the discrete convective terms in Equations (4.4), (4.5), (4.6), and (4.7).

We still need the discrete diffusive terms $\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) \right]_{i,j}$, $\left[\frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \right]_{i,j}$, $\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) \right]_{i,j}$, and $\left[\frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) \right]_{i,j}$, but before discretizing these diffusive terms, we have to discuss how the surface tension terms affect the velocity update in the Equations (4.33) and (4.34). These formulas can be used when away from the interface Γ. However, if the interface exists between $x_{i,j}$ and $x_{i+1,j}$, then the jump in pressure due to surface tension needs to be included on the right hand side of Equation (4.33) as in [3]. The new formula is then

$$u_{i,j}^{n+1} = F_{i,j}^{n} - \frac{\Delta t}{\hat{\rho} \Delta x} \left(p_{i+1,j}^{n+1} - p_{i,j}^{n+1} \right) + \frac{\sigma \kappa}{\hat{\rho} \Delta x}, \tag{4.37}$$

where $\hat{\rho}$ is the averaged density used in the pressure equation for the discrete term $\left[\frac{1}{\rho} \frac{\partial p}{\partial x} \right]_{i+\frac{1}{2},j}$. Similarly, if the interface exists between $x_{i,j}$ and $x_{i,j+1}$, then the jump in pressure due to surface tension is included on the right hand side of Equation (4.34) as

$$v_{i,j}^{n+1} = G_{i,j}^{n} - \frac{\Delta t}{\rho \Delta y} \left(p_{i,j+1}^{n+1} - p_{i,j}^{n+1} \right) + \frac{\sigma \kappa}{\hat{\rho} \Delta y}, \tag{4.38}$$

where $\hat{\rho}$ is the averaged density used in the pressure equation for the discrete term $\left[\frac{1}{\rho} \frac{\partial p}{\partial y} \right]_{i,j+\frac{1}{2}}$.

The discretization of the diffusive terms in Equations (4.35) and (4.36) is done in a similar manner as the discrete pressure terms discussed in Section 4.2.2. We will consider the diffuse terms $\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) \right]_{i,j}$ and $\left[\frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \right]_{i,j}$ from Equation (4.35) first.

We discretize the term $\left[\frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) \right]_{i,j}$ in almost exactly the same way that we discretize $\left[\frac{\partial}{\partial x} \left(\frac{1}{\rho} \frac{\partial p}{\partial x} \right) \right]_{i,j}$. Indeed, the technique is identical to that discussed in Section 4.2.2, except that instead of the variable coefficient $\frac{1}{\rho}$ we have variable coefficient μ, and our discretization is now centered at $x_{i+\frac{1}{2},j}$ instead of $x_{i,j}$. Because of the similarities with the pressure solve, we can avoid a lengthy discussion of the how to derive the formulas for μ on the left and right arms of the stencil at $x_{i+\frac{1}{2},j}$. The discretizations will take the forms

$$\left[\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)\right]_{i,j} = \frac{\left[\mu\frac{\partial u}{\partial x}\right]_{i+\frac{1}{2},j} - \left[\mu\frac{\partial u}{\partial x}\right]_{i-\frac{1}{2},j}}{\Delta x}, \tag{4.39}$$

and

$$\left[\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\right]_{i,j} = \frac{\left[\mu\frac{\partial u}{\partial y}\right]_{i,j+\frac{1}{2}} - \left[\mu\frac{\partial u}{\partial y}\right]_{i,j-\frac{1}{2}}}{\Delta y}. \tag{4.40}$$

The subscripts above may seem confusing. The term $\left[\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)\right]_{i,j}$ is the discretization for the cell $c_{i,j}$, but it is centered about the point $x_{i+\frac{1}{2},j}$. Hence $\left[\mu\frac{\partial u}{\partial x}\right]_{i+\frac{1}{2},j}$ is actually centered about $x_{i+1,j}$, and $\left[\mu\frac{\partial u}{\partial x}\right]_{i-\frac{1}{2},j}$ is centered about $x_{i,j}$. On the other hand, $\left[\mu\frac{\partial u}{\partial y}\right]_{i,j+\frac{1}{2}}$ is centered about $x_{i+\frac{1}{2},j+\frac{1}{2}}$ and $\left[\mu\frac{\partial u}{\partial y}\right]_{i,j-\frac{1}{2}}$ is centered about $x_{i+\frac{1}{2},j-\frac{1}{2}}$.

Let the viscosity of the air be denoted μ_A and the viscosity of the liquid be μ_L. The discretization of $\left[\mu\frac{\partial u}{\partial x}\right]_{i+\frac{1}{2},j}$ is then

$$\left[\mu\frac{\partial u}{\partial x}\right]_{i+\frac{1}{2},j} = \hat{\mu}\frac{u_{i+1,j} - u_{i,j}}{\Delta x} \tag{4.41}$$

where

$$\hat{\mu} = \begin{cases} \mu_A, & \text{if } \phi_{i+\frac{1}{2},j} > 0 \text{ and } \phi_{i+\frac{3}{2},j} > 0 \\ \mu_L, & \text{if } \phi_{i+\frac{1}{2},j} \le 0 \text{ and } \phi_{i+\frac{3}{2},j} \le 0 \\ \frac{\mu_L\mu_A}{\mu_A\theta+\mu_L(1-\theta)}, & \text{if } \phi_{i+\frac{1}{2},j} \le 0 \text{ and } \phi_{i+\frac{3}{2},j} > 0 \\ \frac{\mu_L\mu_A}{\mu_L\theta+\mu_A(1-\theta)}, & \text{if } \phi_{i+\frac{1}{2},j} > 0 \text{ and } \phi_{i+\frac{3}{2},j} \le 0 \end{cases}$$

and $\phi_{i+\frac{1}{2},j} = \frac{\phi_{i,j}+\phi_{i+1,j}}{2}$, $\phi_{i+\frac{3}{2},j} = \frac{\phi_{i+1,j}+\phi_{i+2,j}}{2}$, and $\theta = \dfrac{\phi_{i+\frac{1}{2},j}}{\phi_{i+\frac{1}{2},j} - \phi_{i+\frac{3}{2},j}}$. Notice that

$$\frac{\mu_L\mu_A}{\mu_A\theta + \mu_L(1-\theta)} = \frac{1}{\frac{1}{\mu_L}\theta + \frac{1}{\mu_A}(1-\theta)},$$

and

$$\frac{\mu_L \mu_A}{\mu_L \theta + \mu_A(1-\theta)} = \frac{1}{\frac{1}{\mu_A}\theta + \frac{1}{\mu_L}(1-\theta)},$$

which highlights that this value of μ is derived in the same manner as ρ in the pressure equation in Section 4.2.2.

The only real difference between the formulas for $\hat{\mu}$ and $\hat{\rho}$ is that we use $\phi_{i+\frac{1}{2},j}$ and $\phi_{i+\frac{3}{2},j}$, which are values of ϕ linearly interpolated at the midpoint of right cell edges, rather than values of ϕ at cell centers. This is because we need values of ϕ located at the same position as $u_{i,j}$ and $u_{i+1,j}$. Similarly, for the discrete term $\left[\mu \frac{\partial u}{\partial x}\right]_{i-\frac{1}{2},j}$, we linearly interpolate values of ϕ at the positions of $u_{i,j}$ and $u_{i-1,j}$, and then the resulting discretization formula is derived in a symmetric manner as above.

For the discretization in Equation (4.40), we derive the discrete terms on the right hand side in exactly the same manner as discussed above for Equation (4.39). In this case we linearly interpolate ϕ to get values located at the same positions as $u_{i,j-1}$, $u_{i,j}$, and $u_{i,j-1}$. Because of this symmetry, the details are omitted.

The diffuse terms from Equation (4.36) are treated in a manner that is symmetric to the diffuse terms from Equation (4.35) just discussed, except that now the discretizations are centered at the midpoints of top cell edges. Because of this, the term $\left[\frac{\partial}{\partial x}\left(\mu \frac{\partial v}{\partial x}\right)\right]_{i,j}$ from (4.36) corresponds more closely to the term $\left[\frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right)\right]_{i,j}$ from Equation (4.35), and the term $\left[\frac{\partial}{\partial y}\left(\mu \frac{\partial v}{\partial y}\right)\right]_{i,j}$ from Equation (4.36) corresponds more closely to the term $\left[\frac{\partial}{\partial x}\left(\mu \frac{\partial u}{\partial x}\right)\right]_{i,j}$ from Equation (4.35). Again, due to symmetry, we omit the details.

4.2.4 Summary of Fluid Simulation Algorithm

The entire algorithm for simulating the two phase flow is given in **Algorithm 1**. In order to analyze the results of the fluid simulation, data about the velocities, pressure, level set, or any parameters used will need to be saved and stored during the simulation. These details are implementation dependent and so left out of our discussion.

Algorithm 1 This is the pseudocode describing how to put all the parts of the fluid solver together.

```
1: Set t:=0, n:=0
2: Assign initial values to $u$, $v$, $p$, and $\phi$.
3: while $t < T$ do
4:     Select the timestep $\Delta t$ according to (4.15).
5:     Set boundary values for $u$ and $v$. For no-slip conditions use (4.9) and (4.10). For free-slip conditions use (4.11) and (4.12).
6:     Compute $F^n$ and $G^n$ according to (4.35) and (4.36).
7:     Compute the right hand side of the pressure equation (4.18) using (4.30). At the interface $\Gamma$, add the surface tension terms described in Section 4.2.2 as well.
8:     Solve the pressure equation as discussed in Section 4.2.2 for pressure $p^{n+1}$.
9:     Compute $u^{n+1}$ and $v^{n+1}$ using (4.33) and (4.34). At the interface $\Gamma$, add the surface tension terms described in Section 4.2.3.
10:    Advect the level set using an upwinding scheme as described in Section 3.3 for one time step of size $\Delta t$.
11:    Reinitialize the level set function to a signed distance function as described in Section 3.3.1.
12:    $t \leftarrow t + \Delta t$, $n \leftarrow n + 1$.
13: end while
```

Chapter 5

Results

Our simulations focused on what we will call the *drop problem*. That is, we were interested in simulating a drop of liquid falling through air and into a pool of the same type of liquid. We are more interested in the liquid than the surrounding air, so the parameters for the air phase are never changed. We will restrict our analysis to the effects of modifying the parameters of the fluid phase. The parameters we have control of are the density ρ, viscosity μ, and surface tension coefficient σ. As a default, all parameters will be set to those of water. So for example, if we are analyzing viscosity, then we will set density and surface tension forces to those of water. But first, before changing the parameters of the liquid from those of water, we will do a basic convergence and qualitative analysis of the simulation with water parameters.

5.1 Convergence Analysis

Our first water drop simulation takes place in a 1×2 m rectangular region with free-slip boundary conditions. The radius of the water drop is 0.15 m and the pool of water below is 1 m high. We use physical parameters for the water and air phases provided in [8]. The density of water is $\rho_{water} = 999.9\ kg/m^3$, and the viscosity is $\mu_{water} = 1.787 \cdot 10^{-3}\ kg/(m\ s)$. For air the density is $\rho_{air} = 1.293\ kg/m^3$, and the viscosity is $\mu_{air} = 1.71 \cdot 10^{-5}\ kg/(m\ s)$. For the surface tension coefficient of water we use the value $\sigma = 0.073\ kg/s^2$ found in [3]. Two more parameters which need to be set are the values of u_∞ and L. However, since u_∞ and L only appear in Equations (2.5) and (2.6) as the product $u_\infty L$, we effectively just need to set the value of $u_\infty L$ directly. Modifying the value of $u_\infty L$ effectively modifies the value of the Reynolds number $Re = \rho u_\infty L/\mu$. We found that the best results were obtained with $u_\infty L = 0.001\ m^2/s$ and use this as a default value, but we will explore this parameter as well.

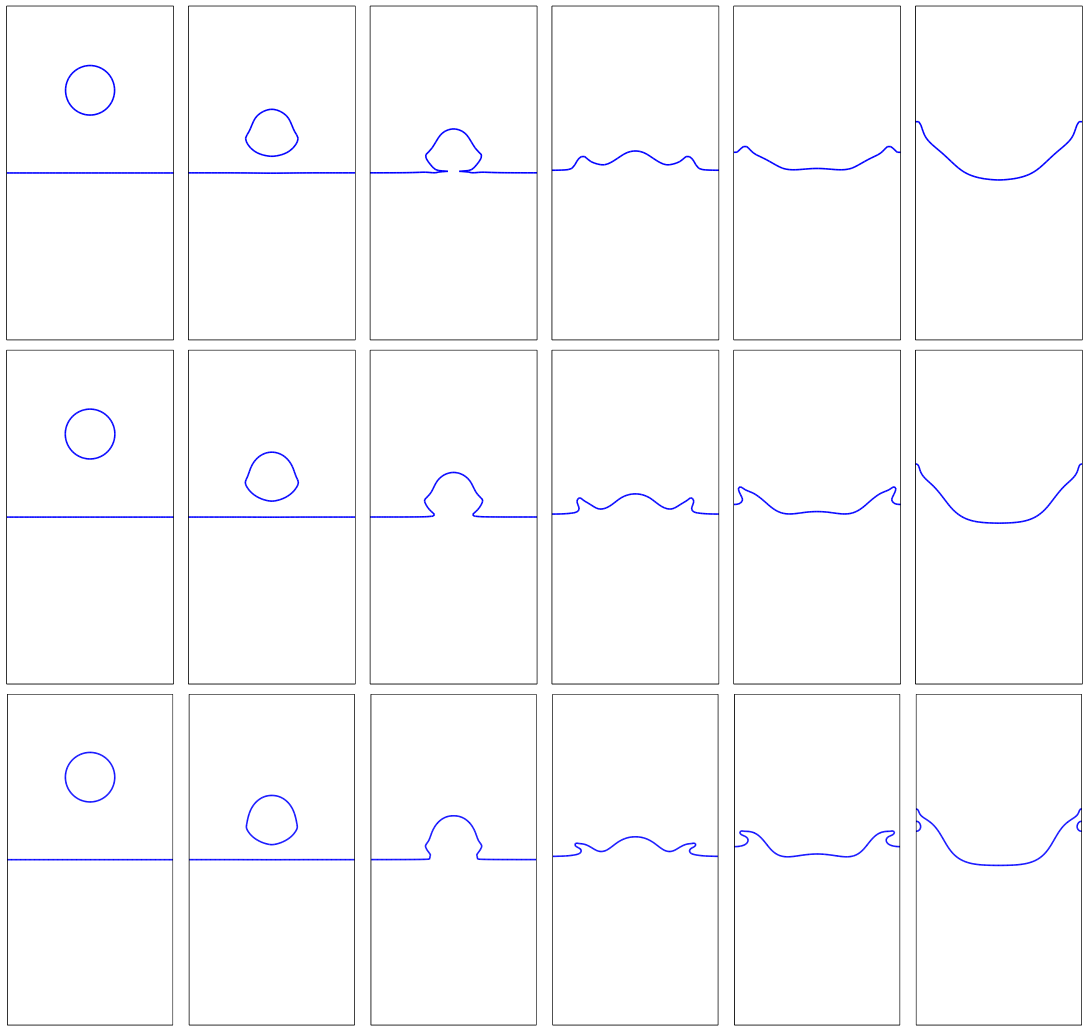

Figure 5.1: Time evolution (from left to right) of the drop simulation with grid sizes 50×100 (top row), 100×200 (middle row), and 200×400 (bottom row). The times t of the frames are (from left to right), $t = 0$, $t = 0.233$, $t = 0.283$, $t = 0.333$, $t = 0.383$, and $t = 0.433$.

We first will look at the behaviour of the simulation for different grid sizes $n_x \times n_y$. In all simulations we set $\Delta y = \Delta x$. Figure 5.1, shows the time evolution of the drop simulation with grid sizes 50×100, 100×200, and 200×400. As the grid becomes more refined (when Δx decreases), finer details of the splash are captured. This is particularly pronounced in the fourth and fifth columns, as the waves which result from the splash have more detailed crests. The improvement in quality is matched by a significant increase in computational time. We compare computational times in Table 5.1. The time step restriction $\Delta t \leq \frac{Re}{2}\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)^{-1}$ implies

Δx (m)	Computation time for one time step (s)	Timestep Δt (s)
1/50	0.4	0.0019
1/100	3.7	0.0009507
1/200	45.8	0.0002377

Table 5.1: A comparison of the typical time taken to evolve the simulation for one time step and the typical size of a time step Δt for different grid sizes.

that we have $\Delta t \propto \Delta x^2$ when $\Delta x = \Delta y$. When Δx is halved from 1/50 to 1/100 we notice that Δt is approximately halved as well, which does not fit the proportionality relationship $\Delta t \propto \Delta x^2$. But, when Δx is halved from 1/100 to 1/200 we notice that Δt is approximately reduced by a factor of four, which does follow the relationship $\Delta t \propto \Delta x^2$, suggesting that this relationship becomes more accurate as mesh size Δx decreases.

The bottleneck in computational time is the implicit pressure solver. While the explicit velocity update will take longer with more grid points, the update is effectively proportional to the number of grid points. Solving for pressure using CG algorithm ends up taking more iterations as Δx decreases, and each iteration needs to consider more grid points. Hence, the time taken per time step increases dramatically as Δx increases. While CG is expected to require more iterations with decreasing Δx, another reason may be that with the sharp interface conditions, the matrix system defined by the *variable coefficient Poisson equation* becomes ill-conditioned. If this is the case, then as more grid points border the interface Γ due to decreasing Δx, the more ill-conditioned the system may become. This claim has not been verified, though such an issue may be partly overcome by employing the PCG or MG methods [14]. Even if our claim is not true, these algorithms would still be expected to improve the pressure solver.

5.2 Surface Tension

In the default simulation, our surface tension coefficient is $\sigma = 0.073$, and the drop has a radius of 0.15 m. At this scale, the effects of surface tension are not very pronounced, as would be expected for such a large body of water. In fact, our simulations with $\sigma = 0$ were essentially identical to those with $\sigma = 0.073$. In order to see the effects of surface tension, we need to increase σ. We explore the parameter σ in Figure 5.2. We can see that the general effect of surface tension is to smooth out the interface Γ. This is the expected effect of surface tension. What is somewhat surprising is the extreme coefficient σ needed to observe these effects. In [3] the authors note that

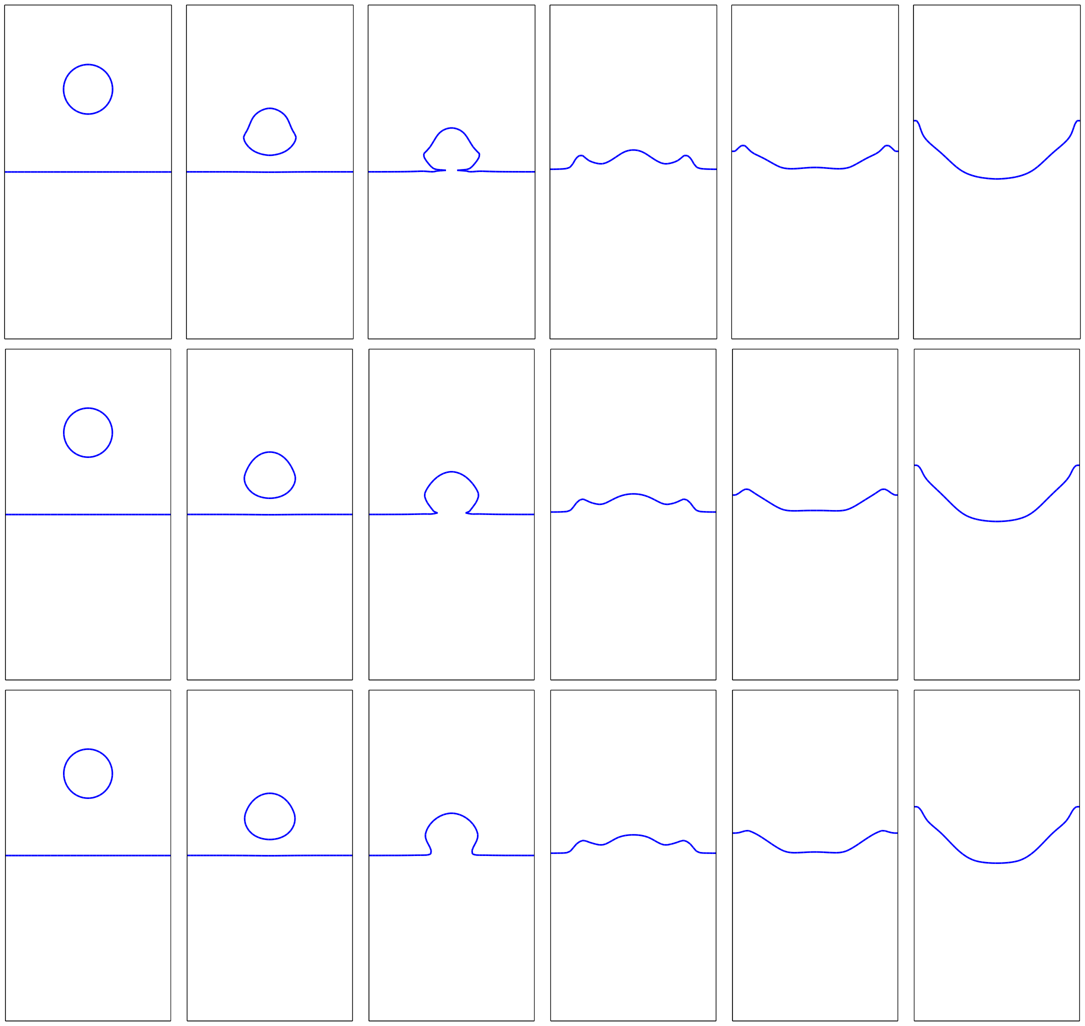

Figure 5.2: Time evolution (from left to right) for different surface tension coefficients σ. First we have default surface tension $\sigma = 0.073\ kg/m^2$ in row one, followed by surface tension $\sigma = 2000.0\ kg/m^2$ in row two, and finally surface tension $\sigma = 4000.0\ kg/m^2$ in row three. The times t of the frames are (from left to right), $t = 0$, $t = 0.233$, $t = 0.283$, $t = 0.333$, $t = 0.383$, and $t = 0.433$

$\sigma = 0.073$ is enough to keep a drop together and rounded. However, their simulations took place on the scale of centimetres, whereas our simulation is on the scale of metres. This helps explain why such an extreme value of σ is needed to significantly affect the simulation.

5.3 Viscosity

The viscosity of the liquid affects how "thick" the fluid appears to be. In Figure 5.3, we compare low, default, and high values of viscosity, which corresponds to thin, default (water), and thick fluid respectively. For thin fluid ($\mu = 0.00001787$), we can see that the crests of the splash are flatter, suggesting that the thin fluid stuggles to push through the air. For thick fluid ($\mu = 0.1787$), the crests of the splash are also suppressed but to a more extreme degree, which is to be expected. When a fluid thickens, it is harder for kinetic energy to carry through it since friction between particles is much higher. For an intuitive (but extreme) example, we would not expect a drop of honey to create a large splash in a pool of honey.

5.4 Density

Here we will modify the density ρ of the liquid phase. In essence this should affect the weight of the liquid. For our simulation, we should expect the lighter liquid to be more heavily influenced by the surrounding air medium. Results are shown in Figure 5.4. We can see that for low density ($\rho = 50.0$) the drop becomes more ovular as it approaches the pool below, suggesting the air resists the falling drop more significantly and deforms it. Also, the pool below deforms well before the drop reaches it. This means that as the air does not move out of the way of the drop as easily as for higher density liquids, and as the drop falls the air pushes on the pool below. Also, since the pressure ratio between the liquid and air is much lower, the drop does not fall as quickly. This effect is small but noticeable in column three of Figure 5.4. This is to be expected since as the pressure ratio approaches $1 : 1$ we would not expect the drop to fall at all.

In the case of the "high" density drop ($\rho = 1500.0$), we do not notice a significant difference in the behaviour of the drop. We were unable to make a more extreme density such as $\rho = 2000.0$ since these simulations exhibited nonphysical behaviour when the drop hit the pool below and the time steps also decreased dramatically. It appears that our simulation stops functioning correctly for extreme pressure ratios. A potential reason for this is that the matrix system for the pressure solve may become ill-conditioned due to the special conditions imposed at the interface Γ. If this were true we would expect the matrix system to become even more poorly conditioned with higher density ratios (compared with a 1:1 ratio which would give a standard, well-behaved Laplacian system).

5.5 Effect of the Product $u_\infty L$

As stated before, we set $u_\infty L = 0.001$ in our simulations as it seemed to provide the most physically believable simulations. But this is an arbitrary choice and so we will briefly explore

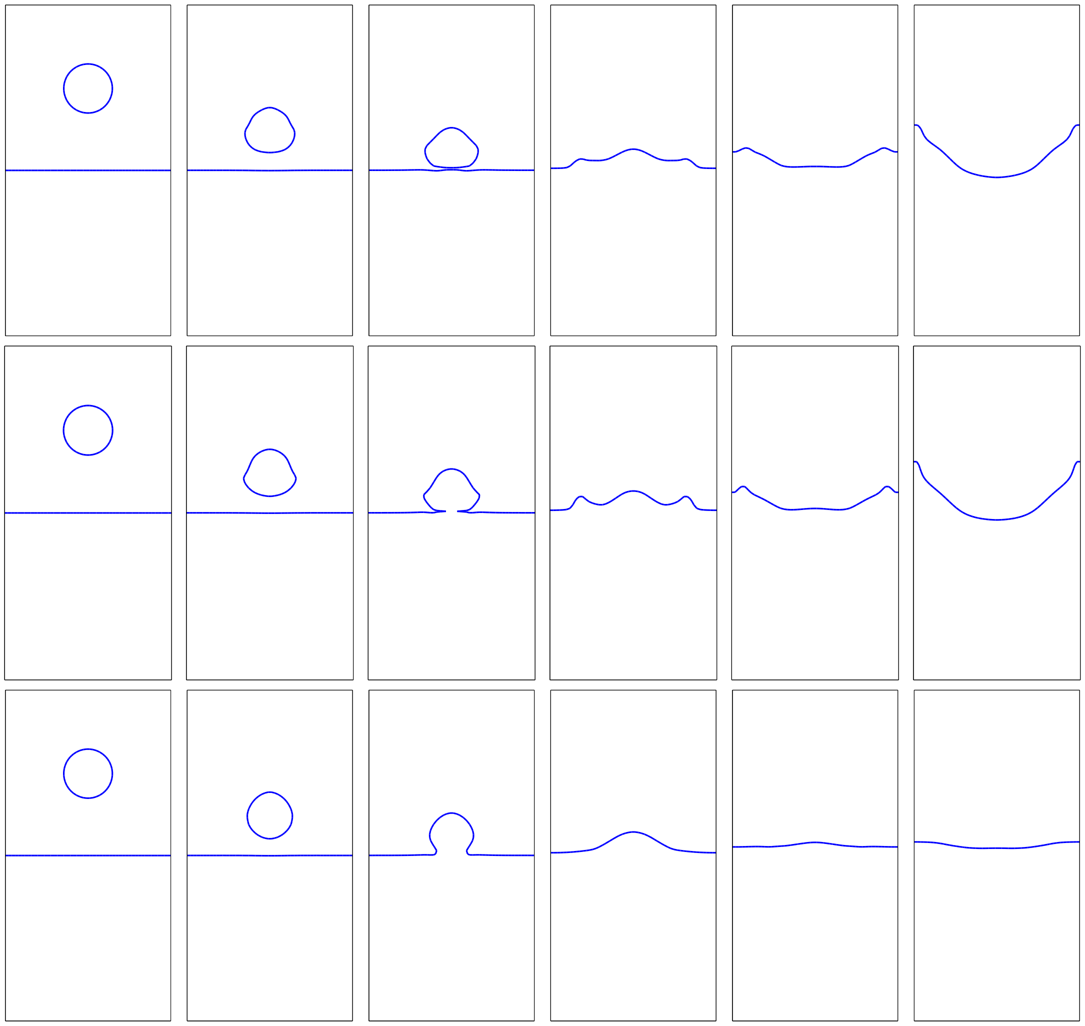

Figure 5.3: Time evolution (from left to right) for different viscosities μ. First we have $\mu = 0.00001787\ kg/(m\ s)$ in row one, followed by $\mu = 0.001787\ kg/(m\ s)$ in row two, and finally $\mu = 0.1787\ kg/(m\ s)$ in row three. The times t of the frames are (from left to right), $t = 0$, $t = 0.233$, $t = 0.283$, $t = 0.333$, $t = 0.383$, and $t = 0.433$

some other values of $u_\infty L$. Figure 5.5 shows the time evolution for various values of $u_\infty L$. We focus on the falling portion of the drop simulation in Figure 5.5 because the subsequent splash and sloshing of liquid does not change significantly for the various $u_\infty L$ values. Due to the scale of our simulation, the default surface tension is nearly nonexistent, making our simulation closer

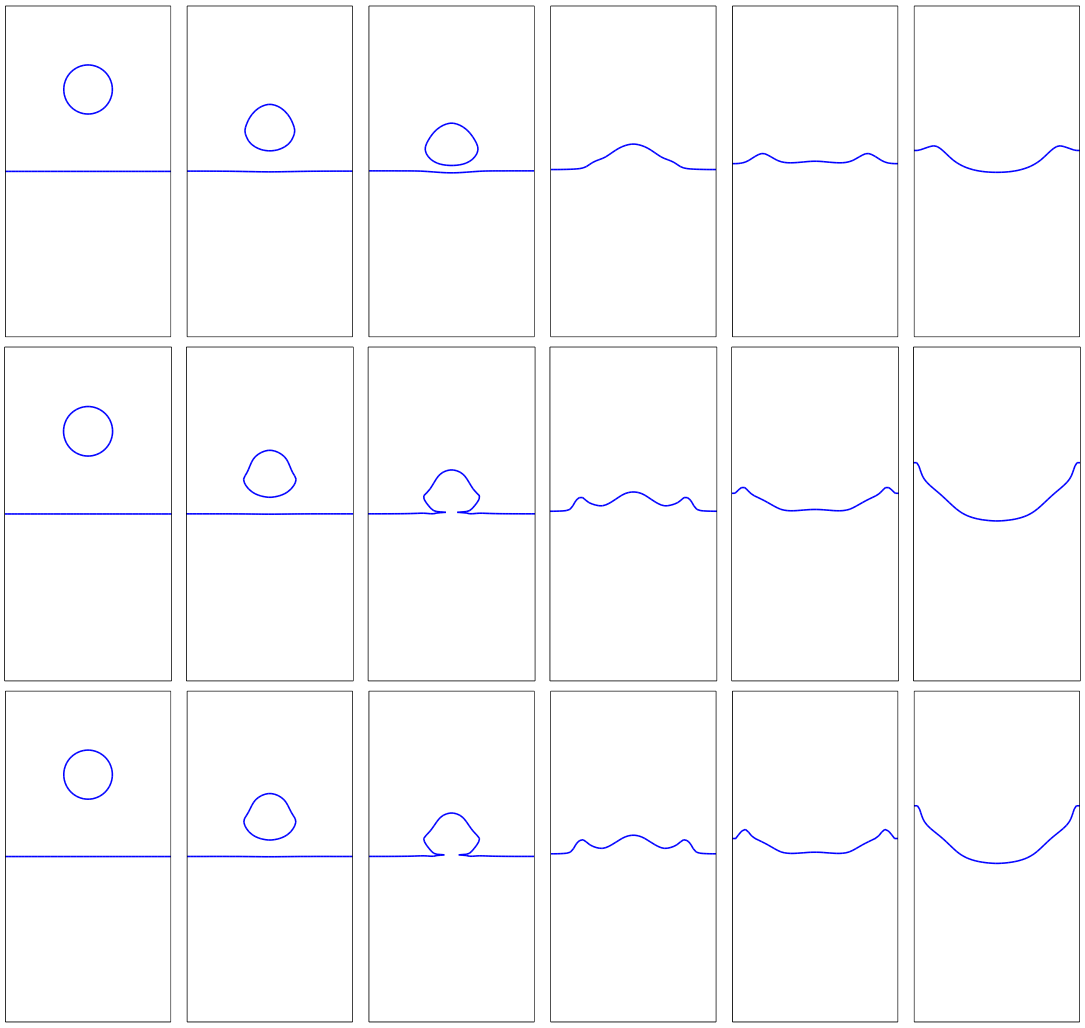

Figure 5.4: Time evolution (from left to right) for different densities ρ. First we have $\rho = 50.0\ kg/m^3$ in row one, followed by $\rho = 999.9\ kg/m^3$ in row two, and finally $\rho = 1500\ kg/m^3$ in row three. The times t of the frames are (from left to right), $t = 0$, $t = 0.233$, $t = 0.283$, $t = 0.333$, $t = 0.383$, and $t = 0.433$

to the drop simulations from [19] with no surface tension. In their simulations, we see a shape like the one in row two, column five of Figure 5.5 resulting from the air resisting the falling drop. Because of this agreement with the results in [19], we chose to keep $u_\infty L = 0.001$.

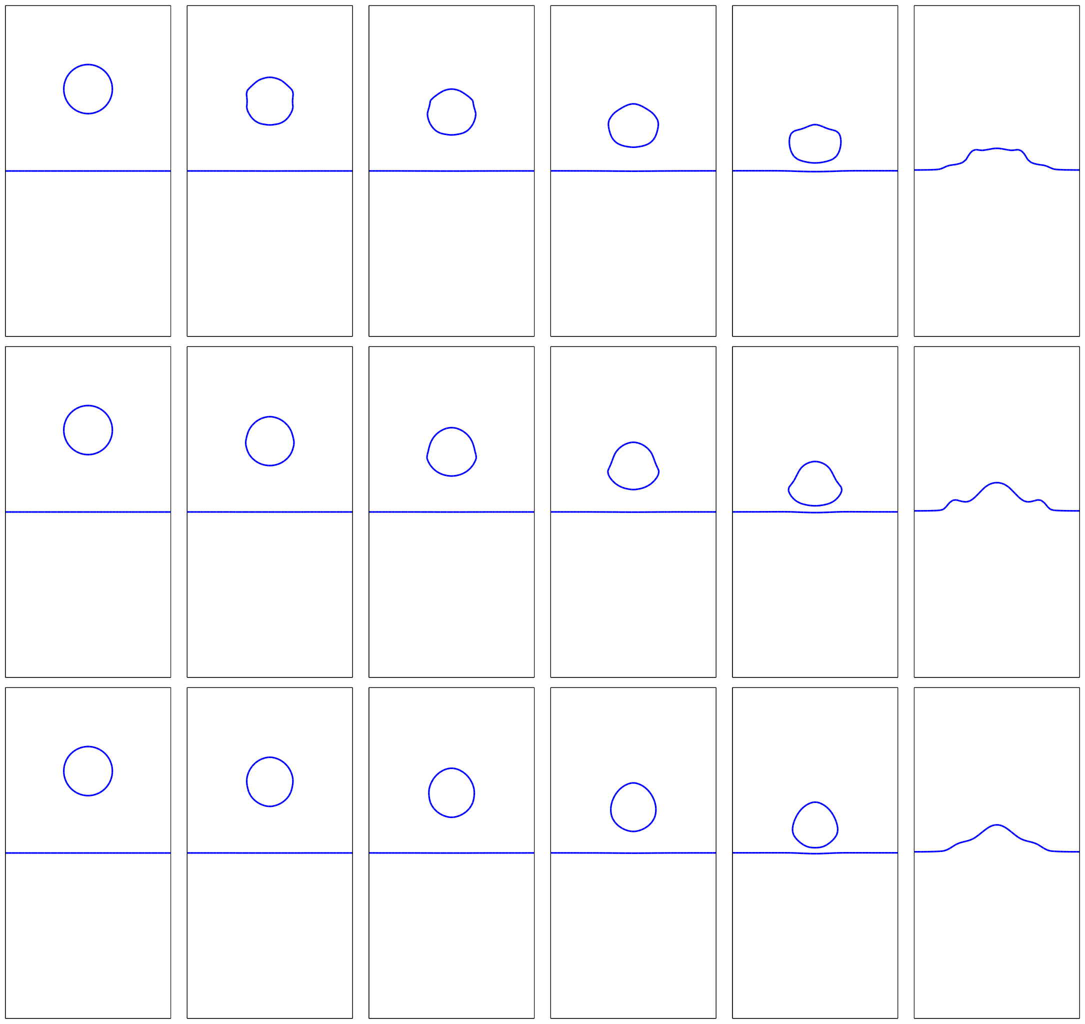

Figure 5.5: Time evolution (from left to right) for different values of $u_\infty L$. First we have $u_\infty L = 0.1\ m^2/s$ in row one, followed by $u_\infty L = 0.001\ m^2/s$ in row two, and finally $u_\infty L = 0.0001\ m^2/s$ in row three. The times t of the frames are (from left to right), $t = 0$, $t = 0.117$, $t = 0.167$, $t = 0.217$, $t = 0.267$, and $t = 0.317$

5.6 Volume Loss

A common problem in two phase flow simulations is volume loss. We can see the issue discussed in the related literature [4, 7, 17, 19, 20]. For methods using level sets to track the interface between

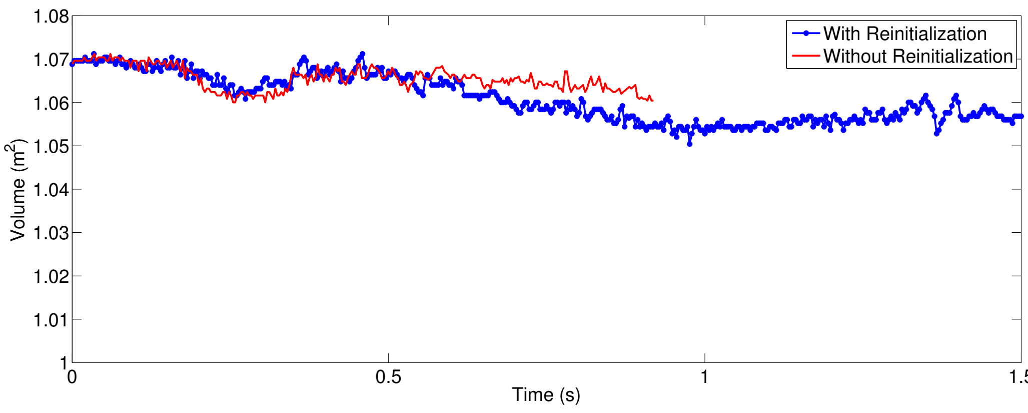

Figure 5.6: This graph tracks the volume of the drop of fluid with and without reinitialization. Without reinitialization the simulation becomes numerically unstable after roughly 0.9 seconds.

the fluids, it is common to partially remedy the volume loss by reinitializing the level set to a signed distance function, as discussed in Chapter 3. The underlying fluid velocity used to advect the level set function ϕ is also supposed to be divergence free, and hence volume loss observed in simulations is a numerical artifact. This means that improvements to the underlying numerical methods should aid in preventing volume loss. An example of this is to improve the forward and backward differences used in the upwinding scheme used to advect the level set function using third order ENO differences (this is discussed in the Appendix). Another way to improve the numerical method is of course to decrease the mesh size Δx.

Figure 5.6 shows a volume comparison of the drop simulation from 0 to 1.5 seconds with and without level set reinitialization. Without reinitialization, the simulation became numerically unstable after roughly 0.9 seconds. This makes a meaningful comparison of the volume loss difficult. However, we can see that the volume in the early simulation is not dramatically different with and without reinitalization. It seems that the major benefit of reinitialization is improved stability more so than prevention of volume loss.

In Figure 5.7 we see a volume comparison for the drop simulation with and without 3rd order ENO approximations. The "dropping" portion of the simulation (which roughly spans the first 0.3 seconds) shows improved volume preservation, whereas the subsequent splash and sloshing of the fluid shows no improvement either way.

We compare volume preservation for two mesh sizes Δx in Figure 5.8. With the finer mesh ($\Delta x = 1/200$) we notice that the volume is more stable over time compared with the volume from the larger mesh ($\Delta x = 1/50$). During the "dropping" portion of the simulation (roughly from $t = 0$ to $t = 0.3$) both mesh sizes show volume loss but the smaller mesh size shows much

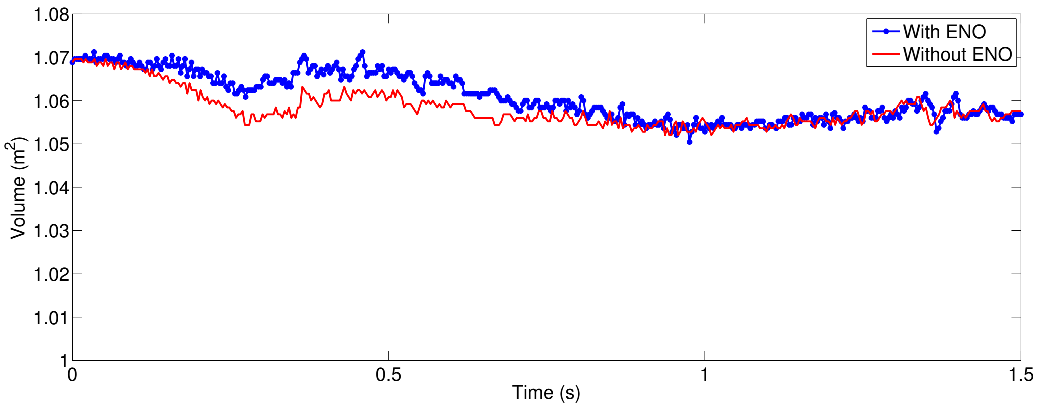

Figure 5.7: This graph tracks the volume of the drop of fluid with and without ENO approximations.

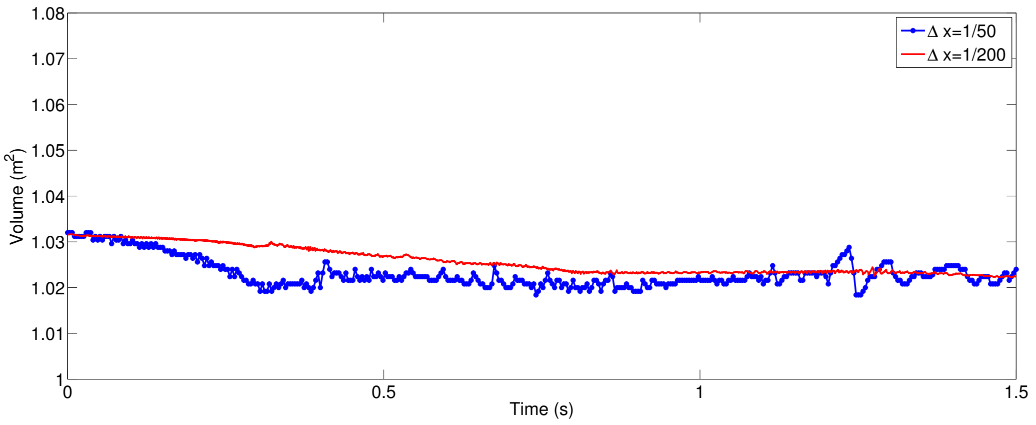

Figure 5.8: This graph compares volume preservation of a drop of radius 0.10 m using a mesh size of $\Delta x = 1/50$ (blue) and $\Delta x = 1/200$ (red).

less, and is more stable. In the subsequent splash both simulations slow their volume loss and in fact end up with a similar volume at time $t = 1.5$. The stability of volume exhibited with the smaller mesh size is of course more desirable than the comparatively volatile volume shown with the larger mesh size, as it corresponds more closely to the principle of conservation of mass used to derive the continuity Equation (2.3).

Chapter 6

Conclusion

Here we wish to briefly reflect on several aspects of our fluid solver. We will begin in Section 6.1 with a discussion of technical issues of our simulation and possible improvements. This is followed by a brief look at potential directions for future work in Section 6.3. For readers looking to implement the fluid simulator described in Chapter 4, we will share some of the lessons learned from our implementation experience in Section 6.2.

6.1 Technical Issues

There were a number of technical problems with the results of our simulations. We will discuss some of them here, looking to related literature for potential solutions.

One notable issue was the problem of volume loss. One volume preservation measure (discussed in Section 5.6) was the use of third-order ENO approximations while advecting the level set function ϕ. We also discussed the reinitialization of ϕ to a signed distance function as a volume preservation tool. Both measures partly addressed volume loss, but reinitialization seemed to be more helpful for stability. Another measure to preserve volume was to reduce the mesh size Δx at the cost of a dramatic increase in simulation time. If we look in Figure 5.1 at the panel in row three and column six, we can see two small bubbles formed along the left and right walls at the peak of the waves. We found that small bubbles such as these would not survive long in our simulations. We also found that drops with a small enough initial radius would not survive long enough to reach the pool of liquid below. Similarly, we were not able to observe any small drops of water breaking away from the drop's splash.

Our simulations treat the interface in a somewhat simplistic manner, mostly borrowing the sharp interface techniques employed in [7] when solving for pressure and discretizing the diffusive terms of Equations (2.5) and (2.6). Other sharp interface techniques treat the interface more carefully. In some methods, separate fluid velocities for each phase are used and combined to advect the level set (as in [3, 7]). Another common technique is to use the motion of particles at the interface Γ to more accurately capture the motion of the interface [3, 5]. Another technique is the *Coupled Level Set and Volume-of-Fluid* (CLSVOF) method which can be found in [3, 18]. In short, many other two phase fluid simulators recognize and address the issue of volume loss at the interface Γ using more advanced techniques than those used in our simulations. It would certainly be worth exploring and implementing any of the methods above to help improve our simulation's treatment of the interface Γ.

We mentioned that reducing Δx improved volume preservation at the cost of a much longer simulation time. We also discussed in Section 5.1 that the matrix system defined by the *variable coefficient Poisson equation* may become ill-conditioned due to the conditions imposed at the interface Γ. It is possible that as more grid points neighbour the interface Γ, due to a smaller grid size Δx, that the system becomes more ill-conditioned. If this is the case, then an obvious technical improvement would be to improve the pressure solver used. In our simulations, we used the CG algorithm which might have trouble converging for an ill-conditioned system. Perhaps the use of PCG, which should help to improve the conditioning of the matrix system, or MG, which takes far fewer iterations to converge, could improve the simulation time of our algorithm [14]. This would also allow us to reduce Δx, and hence, improve the accuracy of our simulations without such a large penalty in simulation time.

Another potential improvement to our simulation is to use a more advanced time discretization, such as Runge-Kutta [17, 19]. We used a forward Euler time discretization in our simulations which is one of the most basic techniques. This may have to be carefully considered since our time stepping loop is explicit in velocity and implicit in pressure. Hence, implementing a Runge-Kutta time discretization may not be completely straightforward.

6.2 Notes on Implementation

We have implemented our fluid simulator in C++ exactly as described in Chapter 4. However, our final simulation is in two dimensions whereas we started with a three dimensional simulation. We began with a simulation of a single fluid phase in a closed box and then tried to transform this into a two-phase flow simulation. Unfortunately, we ran into bugs as is to be expected. Finding and fixing those bugs in three dimensions was very time consuming since the code was much longer than the current two-dimensional implementation, and also because the three dimensional simulations took much longer to compute. It was because of these time constraints that we had to scale back our simulations to two dimensions to work out the bugs.

For any readers looking to implement a fluid simulation, either as described in Chapter 4 or otherwise, we suggest beginning with a two dimensional simulation. Only once the bugs in the two dimensional simulations are worked out, should three dimensional simulations be attempted. In [8] one can find the three-dimensional equations of motion, but they are simple extensions of the two dimensional equations of motion. Hence they do not add any fundamentally new challenges to the implementation.

Visualization can be a challenge in three dimensions. In two dimensions, we used MATLAB's *contour* command to extract the level set of ϕ after writing the contents of ϕ in a MATLAB file format. In three dimensions, one can perform a similar level set extraction using the *isosurface* command. However, MATLAB visualization can be limiting, and readers may wish to extract a mesh of the level set of ϕ themselves using the well-known *marching cubes* or *marching tetrahedrons* algorithm. This allows the freedom to visualize the mesh using ray tracing or rasterization techniques. Before scaling back to two dimensions, we used the marching tetrahedrons algorithm for mesh extraction and a ray tracer for visualization.

6.3 Future Work

We will only briefly discuss potential avenues for future work. Besides implementing any of the technical improvements in the previous section, an obvious direction for future work is an extension to three dimensions. In fact, we originally attempted a three dimensional simulation before scaling back to two dimensions to work out bugs in our program in a more time efficient manner. Because the derivative approximations and level set techniques used in our simulation essentially consider the x and y dimensions separately, there is nothing fundamentally difficult about extending the algorithm to three dimensions. In component form, the three dimensional momentum equations are now three separate equations instead of two, each corresponding to an axis-oriented velocity. A few extra terms show up in the derivation, but they are predictable. A brief discussion of three dimensional simulations can be found in [8], the book upon which most of our discretizations were based.

Another direction for future work is to study non-Newtonian fluids such as human blood. This requires at least a modification of the stress tensor $\boldsymbol{\sigma}$, which was derived based on the assumption of a Newtonian fluid. For blood in particular, the stress tensor should be modified to account for the interaction of red blood cells and plasma (remember that the stress tensor captures internal friction forces). Another option would be to model red blood cells directly, perhaps as semi-solid membranes containing fluid, and simulate these blood cells within plasma. This would require solid-fluid coupling techniques as in [1, 12].

The fluid solver as presented should be relatively easy to implement for interested readers. We hope that this work can act as a stepping stone for non-experts looking to implement a first time fluid solver. We have seen in Section 6.1 that a variety of advanced techniques exist for level set methods, and we believe our fluid solver should be general enough to explore these advanced techniques in future work.

APPENDICES

Appendix A

Essentially Non-Oscillatory Derivative Approximations

Here we will discuss how to find *essentially non-oscillatory* (ENO) derivative approximations to a function ϕ. The derivatives in the x and y directions are approximated symmetrically and so we restrict the discussion to a one dimensional function ϕ.

We follow closely the description in [6] and use the smoothest possible polynomial interpolation of ϕ across several grid points and then differentiate to get ϕ_x. Using standard Newton polynomial interpolation terminologies, the zeroth divided differences of ϕ are defined at the cell centers and defined by,

$$D_i^0\phi = \phi_i, \tag{A.1}$$

for each cell center. The first divided differences of ϕ are defined at cell edges (midway between cell centers) as,

$$D^1_{i+\frac{1}{2}}\phi = \frac{D^0_{i+1}\phi - D^0_i\phi}{\Delta x}, \tag{A.2}$$

given a grid size Δx. Notice that $D^1_{i-\frac{1}{2}}\phi$ is just a backward difference of ϕ and $D^1_{i+\frac{1}{2}}\phi$ is a forward difference of ϕ. The second divided differences are defined at the cell centers as,

$$D_i^2\phi = \frac{D^1_{i+\frac{1}{2}}\phi - D^1_{i-\frac{1}{2}}}{2\Delta x}. \tag{A.3}$$

Finally, the third divided differences are defined at cell edges as,

$$D^3_{i+\frac{1}{2}}\phi = \frac{D^2_{i+1}\phi - D^2_i\phi}{3\Delta x}. \tag{A.4}$$

The divided differences are used to construct a polynomial approximation of ϕ,

$$\phi(x) \approx Q_0(x) + Q_1(x) + Q_2(x) + Q_3(x) \tag{A.5}$$

to the function ϕ near x_i. The polynomial approximation can be differentiated and evaluated at x_i to find derivative approximations $(\phi_x^-)_i$ and $(\phi_x^+)_i$. That is, we approximate ϕ_x at x_i with,

$$\phi_x(x_i) \approx Q_1'(x_i) + Q_2'(x_i) + Q_3'(x_i). \tag{A.6}$$

Note that $Q_0(x)$ is a constant, and so vanishes after differentiation.

To find ϕ_x^- we let $k = i - 1$, and to find ϕ_x^+ we let $k = i$. Now we define,

$$Q_1(x) = (D^1_{k+\frac{1}{2}}\phi)(x - x_i), \tag{A.7}$$

which gives,

$$Q_1'(x_i) = D^1_{k+\frac{1}{2}}\phi. \tag{A.8}$$

This says that the contribution of $Q_1'(x)$ in Equation (A.5) is the backward difference in the case of ϕ_x^- and the forward difference in the case of ϕ_x^+. If we only included $Q_1'(x)$ on the right hand side of (A.6) then the resulting derivative approximation of ϕ_x would simply be the first order accurate upwinding approximation. Including $Q_2'(x)$ in Equation (A.6) gives a second order approximation, and including $Q_3'(x)$ gives a third order approximation.

So far we have included points x_k and x_{k+1} in our approximation. In defining $Q_2(x)$, we can include the next point to the left and use $D^2_k\phi$, or the next point to the right and use $D^2_{k+1}\phi$. ENO approximations look to avoid interpolating near large variations of ϕ. For this reason we compare $|D^2_k\phi|$ and $|D^2_{k+1}\phi|$. If $|D^2_k\phi| \leq |D^2_{k+1}\phi|$ then we set $c = D^2_k\phi$ and $k^* = k - 1$. Otherwise we set $c = D^2_{k+1}\phi$ and $k^* = k$. Now we can define,

$$Q_2(x) = c(x - x_k)(x - x_{k+1}), \tag{A.9}$$

so that we get,

$$Q_2'(x_i) = c(2(i - k) - 1)\Delta x, \tag{A.10}$$

as the second order accurate correction to the approximation of ϕ_x in Equation (A.6).

We have not yet used k^*, and if we stop here we have a second order accurate approximation of ϕ_x. To make our approximation third order accurate we need to define $Q_3(x)$ from Equation (A.5). Similar to the second order accurate correction, we get the third order accurate correction by comparing $|D^3_{k^*+\frac{1}{2}}\phi|$ and $|D^3_{k^*+\frac{3}{2}}\phi|$. If $|D^3_{k^*+\frac{1}{2}}\phi| \leq |D^3_{k^*+\frac{3}{2}}\phi|$ we set $c^* = D^3_{k^*+\frac{1}{2}}\phi$. Otherwise we set $c^* = D^3_{k^*+\frac{3}{2}}\phi$. Now we define,

$$Q_3(x) = c^*(x - x_{k^*})(x - x_{k^*+1})(x - x_{k^*+2}), \tag{A.11}$$

so that we get,

$$Q_3'(x_i) = c^*(3(i - k^*)^2 - 6(i - k^*) + 2)\Delta x^2, \tag{A.12}$$

as the third order accurate correction to the approximation of ϕ_x in Equation (A.6).

References

[1] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 26(3):100, 2007.

[2] John B. Bell and Daniel L. Marcus. A second-order projection method for variable-density flows. *Journal of Computational Physics*, 101(2):334–348, 1992.

[3] Landon Boyd and Robert Bridson. Multiflip for energetic two-phase fluid simulation. *ACM Transactions on Graphics (TOG)*, 31(2):16, 2012.

[4] Yu-Chung Chang, T.Y. Hou, B. Merriman, and S. Osher. A level set formulation of Eulerian interface capturing methods for incompressible fluid flows. *Journal of computational Physics*, 124(2):449–464, 1996.

[5] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[6] R. Fedkiw and S. Osher. Level set methods and dynamic implicit surfaces. *Surfaces*, 44:77, 2002.

[7] Ronald P. Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152(2):457–492, 1999.

[8] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction*, volume 3. Siam, 1997.

[9] Francis H. Harlow and Jacob E. Fromm. Computer experiments in fluid dynamics. *Scientific American*, 212:104–110, 1965.

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

[11] Xu-Dong Liu, Ronald P. Fedkiw, and Myungjoo Kang. A boundary condition capturing method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 160(1):151–178, 2000.

[12] Avi Robinson-Mosher, Tamar Shinar, Jon Gretarsson, Jonathan Su, and Ronald Fedkiw. Two-way coupling of fluids to rigid and deformable solids and shells. In *ACM Transactions on Graphics (TOG)*, volume 27, page 46. ACM, 2008.

[13] Giovanni Russo and Peter Smereka. A remark on computing distance functions. *Journal of Computational Physics*, 163(1):51–67, 2000.

[14] Yousef Saad. *Iterative methods for sparse linear systems.* Siam, 2003.

[15] James A Sethian et al. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11(1):1–2, 2003.

[16] Ian F. Stewart. Fluid-tracking methods for the animation and rendering of fluids. Master's thesis, University of Waterloo, 2001.

[17] Mark Sussman, Emad Fatemi, Peter Smereka, and Stanley Osher. An improved level set method for incompressible two-phase flows. *Computers & Fluids*, 27(5):663–680, 1998.

[18] Mark Sussman and Elbridge Gerry Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162(2):301–337, 2000.

[19] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational physics*, 114(1):146–159, 1994.

[20] Mark Sussman, Kayne M. Smith, M. Yousuff Hussaini, Mitsuhiro Ohta, and R. Zhi-Wei. A sharp interface method for incompressible two-phase flows. *Journal of computational physics*, 221(2):469–505, 2007.

Index