UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Comparison of Grid-Based Techniques for Navier-Stokes Fluid Simulation in Computer Graphics

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in
Computer Science

by

Cameron Chrisman

Committee in charge:

Professor Henrik Wann Jensen, Chair
Professor Matthias Zwicker
Professor Samuel R. Buss

2008

The Thesis of Cameron Chrisman is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2008

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

ρ	fluid density
μ	dynamic viscosity
ν	kinematic viscosity $= \frac{\mu}{\rho}$
$\mathbf{u}$	the vector field representing fluid velocity
u	the x component of $\mathbf{u}$
v	the y component of $\mathbf{u}$
w	the z component of $\mathbf{u}$
p	the pressure inside the fluid volume
$\mathbf{f}$	the body force field acting on the fluid
$\mathbf{a}$	the vector field representing fluid acceleration
$\Delta x, \Delta y, \Delta z$	grid spacing in the x, y and z dimensions
dt or Δt	timestep
σ	the stress vector as defined in continuum mechanics theory
τ	the strain vector as defined in continuum mechanics theory
ϵ	the displacement vector of fluid volume, also denoted $\xi\mathbf{i} + \eta\mathbf{j} + \zeta\mathbf{k}$
n	the Poisson ratio of the fluid
∇	the gradient operator
∇^2	the laplacian operator
$\nabla\cdot$	the divergence operator
$\frac{D}{Dt}$	the substantial derivative
Φ	the flux limiting function defined as $\Phi(r)$
ϕ	generic variable for a scalar field

ACKNOWLEDGMENTS

I would like to thank my thesis committee, especially my committee chair, Henrik Wann Jensen, for his patience and kind attention working through several drafts. I would also like to thank my other two committee members, Matthias Zwicker and Samuel Buss, for their feedback and useful input while developing my thesis and proofreading my final drafts. Finally, I would like to thank the UCSD Computer Science Department, for supporting and challenging me in both my graduate and undergraduate career.

ABSTRACT OF THE THESIS

A Comparison of Grid-Based Techniques for Navier-Stokes Fluid Simulation in Computer Graphics

by

Cameron Chrisman
Master of Science in Computer Science
University of California, San Diego, 2008
Professor Henrik Wann Jensen, Chair

Fluid simulation is becoming increasingly common in the film industry and in computer games, however, understanding and implementing a numerical solution to the Navier-Stokes equation can be a daunting task. In this thesis we present an introduction to fluid simulation in computer graphics with a full derivation of the compressible and incompressible Navier-Stokes equation from Newton's Second Law, explaining concepts such as *pressure* and *viscosity* as they are derived. We also present a survey and comparison of four different methods for solving the advection equation: a simple 1^{st} order differencing method, a 4^{th} order QUICK scheme, the popular semi-lagrangian method, and the PIC & FLIP particle based methods. We present these advection scheme comparisons in a series simplified 1D channel plots, and a set of 2D fluid animations. Based on these results, we classify algorithms as stable or unstable, and discuss methods to prevent divergence in unstable solvers.

I

Introduction

I.1 Fluid Simulation in Computer Graphics

Physical simulation in computer graphics is becoming an increasingly common, and indispensable, part of the toolset of special effects studios and digital artists, and use of advanced simulation techniques is increasing rapidly both in film production and interactive entertainment. There are several motivations for why one would want to simulate an effect instead of delegating the responsibility to an artist; the two most important are that hand animated visual effects are often unconvincing, or are on a scale large enough that hand animation is prohibitively time consuming. Consider, for example, that an artist may be able to plausibly animate a single rock tumbling down grassy hillside, but animating an entire landslide that must correctly interact with the terrain and scenery would be both tedious and error prone.

Hand animation of fluids generally suffers from both of these problems; the physical behavior of a fluid is complex, and other than in specific situations that we're familiar with, such as waves or ripples, it is very difficult to intuitively predict its behavior. In addition to this, water is a ubiquitous material in nature, and in natural scenes that are the target of the visual effects industry, may be present in very large quantities and high detail, such as a flood or raging river, a heavy rain falling and accumulating in a complex scene, or ocean waves crashing against rocky cliffs.

This thesis focuses primarily on *grid based* fluid simulation techniques. The earliest and most well studied computational methods for simulating fluids involve discretizing the computational domain into discrete cells, forming a grid, and solving for fluid variables for each grid cell. Grid based fluid simulation techniques are popular in many fields, including computer graphics, primarily because they produce consistently accurate and high quality results under a wide variety of simulation settings, and make use of the large body of general discretized methods of solution such as finite differencing and finite elements.

I.2 Previous Work

Computational fluid dynamics is a highly interdisciplinary field, and is very application oriented. As such, previous work in fluid simulation naturally segments itself into advances in fundamental computational methods, and applications and extensions that are specific to a given field. As with many interdisciplinary fields, methods developed with one intention often find applications elsewhere, and are added to the body of 'CFD wisdom' for that field. Here we will present a brief overview of the core computational methods and techniques that have proven useful in computer graphics, and applications or extensions of those techniques developed solely for use in the field of computer graphics.

I.2.A Computational Methods

Before discussing specific works on fluid simulation, it is worthwhile to point out a single paper that has had a dramatic and profound technique on fluid simulation research both in not only computer graphics, but in engineering and physics disciplines as well.

One of the most commonly cited grid-based technique in computer graphics is the *Marker and Cell method* [24], was originally presented in an early non-graphics work by Frank Harlow, an engineer in the T-3 fluid dynamics division at Los Alamos National Labs in 1965. Harlow's work has been some of the most influential in computational fluid dynamics literature, and has pioneered several other techniques.

Briefly, the Marker and Cell, or *MAC*, method consists of two components: markers, that define the where the fluid is present, and grid cells that store physical variables of the fluid, such as the velocity, pressure, temperature, etc. The advantage of this method is that the markers, a particle system for example, are often capable of representing the fluid at a higher level of detail than the spatial grid, which can be relatively coarse.

Harlow's later work also had a profound impact on the CFD community, and other significant works in computational fluid dynamics that have found their way into common usage in computer graphics. In even earlier works Harlow introduced the particle-in-cell or *PIC* method [15], [25] . These works were later extended to produce the more accurate *FLIP* technique by Brackbull et al. in 1986 [27].

Let us now examine some of the early computer graphics research on grid based fluid simulation. Kass et al. [28] is one of the first major papers in graphics in which the Navier-Stokes equations are solved for the purposes of simulating fluid. Their simulation is simplified greatly, assuming only shallow water surfaces, which restricts water to a heightfield, even in 2D . Later, Chen et al. [7] solved the full Navier stokes equations, but due to the performance constraints of the time, this was done only in two dimensions, although an ad-hoc three dimensional representation was generated using a heightfield based on the pressure value in the cells of a 2D grid.

Two major advancements in the usage of CFD in computer graphics came with the work of Nick Foster and Jos Stam. In 1996, Foster [19], presented one of the first papers in graphics to present results for a full three dimensional solution to the Navier-Stokes equation, referencing Harlow original paper. Foster helped promote the concept of arbitrary methods, not just particles, for the markers of the free surface. The originally proposed alternative to particles was heightfields, which have the disadvantage that they are restricted to a single valued function, and not able to represent 'overturning' flows or splashes.

Another major advancement in the fluid simulation in computer graphics was by Jos Stam [44] in which the concept of *semi-lagrangian* advection was introduced to the computer graphics community. In addition to being a simple and robust approach to solving the advection component of the Navier-Stokes equation, Stam's paper contained polished results and introduced and popularized a variety of easy to implement, efficient methods for solving the other aspects of the Navier-Stokes equation. As a result, interest in, as well as overall technical proficiency in fluid simulation techniques of the graphics community increased significantly.

A major advance in surface representation came with Foster & Fedkiw [18] and in which the concept of using level sets in place of heightfields or particles to mark the free surface. Level sets present an attractive alternative to particles and heightfields, in that the surface can be represented in very high detail, as well as desirable secondary characteristics such as surface curvature. Generally even with particle based simulation methods, the final step before a high quality rendering is to extract a smooth surface from the particles. Enright et al. [14] extended this technique by increasing the accuracy of the level set evolution by seeding particles near the level set surface, developing a method that hybridizes the original particle based marker method and the newer level set formulation.

Many methods for simulating the Navier-Stokes equations do not rely on a grid based discretization at all. One such popular technique is known as *smoothed particle hydrodynamics*, or SPH. SPH uses particles, instead of grid cells, to represent infinitesimal volumes of fluid. Each particle has an associated velocity, pressure, temperature, or any quantity that one desires to be present in the fluid, and each quantity has an associated *kernel function*; summing the kernels of nearby particles is used to interpolate at locations between points. In this way, the particles provide a sampling of the fluid as a function of these variables. The main advantage of SPH is that particles, since they have no connectivity, provide a powerful and easy to use framework for adaptively sampling fluid areas of interest, by simply increasing the particle density in the desired area. Particle-only simulations also generally require less memory than grid-based solutions (most of which make use of particles *and* grids).

Originally presented in Lucy, et al. [30], due to its flexibility, SPH has become the method of choice in the astrophysics community. SPH has also become popular in the real-time graphics community, as the lack of a computational grid significantly decreases memory and computational overhead of fluid simulation algorithms, resulting in several publications targeting realtime simulation [10], [36].

Another class of fluid simulation techniques that has recently become popular is the use of the *Lattice Boltzman* method, in which particles travel along the edges of a 3D lattice, whose motion is based on the statistical mechanics of the fluid in question. Lattice-Boltzmann methods are popular because they provide potential for high resolution flows, and easily extend to two-phase coupled flows. Thuerey et al. [37] incorporates Lattice-Boltzmann methods with the popular level-set framework to produce free surface flows.

Additionally there exist a wide variety of methods that do not solve the Navier-Stokes equations at all, and are derived empirically or mathematically. For example Fournier in 1986 [21], one of the earliest fluid simulation papers in computer graphics, provided a procedural formulation for ocean waves based on an early mathematical model of waves by developed Frantisek Gersnter in the early 19th century.

A more advanced technique was employed by Tessendorf [48], in which the discrete inverse Fourier transform of the Kolmogorov spectrum is used to model waves in the open ocean, which provides extremely convincing animation of ocean swells and choppy waves, but requires the assumption of a large and open fluid domain, making it ill suited for general purpose use.

I.2.B Applications in Computer Graphics

Core advances in the fundamental techniques of fluid simulation are important, and often lead to stunning images, but an equally, if not more important area of research is the application and extension of existing techniques and their unique applications to computer graphics. The two most common applications of fluid simulation in computer graphics are the visual simulation of liquids and gases, although other applications are not uncommon.

Hot gases, usually visualized as smoke or steam, have a long history as a motivating application. An early paper by Stam [46] presented the evolution of gas density using the inverse Kolmgorov spectrum, producing turbulent flows, but with similar limitations associated with this method for modeling ocean waves, making it unsuitable for small scale phenomena. Foster [20] presented a technique building upon his earlier work [19] to model gas density evolution on a grid using finite differencing with the Navier-Stokes equation. Later Fedkiw et al. [16] presented both a technique to simulate smoke that preserves lively vortexes, a problem encountered using semi-lagrangian advection solvers, as well as presenting an offline rendering scheme utilizing photon mapping. More recent work by Selle et al. [43] extends the concept of high vorticity simulation methods using vortex particles, based on previous two-dimensional work [32].

While fluid simulation is an extremely useful tool for visualizing things like water or smoke, or viscous goo, the physical effects of fluids play a key role in how they interact with rigid objects. For example Dorsey et al. [12] uses an iterative application of CFD at the surfaces of objects as part of a method to simulate aging and weathering effects. Bridson et al. [53] presented an extension of a general fluid simulation to animating sand using the physics of granular material, and also introduced the PIC and FLIP methods to computer graphics community.

Coupling fluid simulation with rigid bodies has a significant body of work both in computer graphics and other fields [6], [4]. Often times solid-fluid interaction in grid based methods have degenerate cases when the width of the interacting object is on the scale of the individual grid cells. Grid-based methods have also been extended to handle fluid-solid coupling for thin shells (a common problem when discretizing solid objects into a coarse simulation grid) and deformable bodies [23].

A major challenge in fluid simulation is computational efficiency. As naive grid based methods increase in both space and time complexity as $O(N^3)$, attaining a desirable resolution often comes at prohibitive computational cost. Lossaso et al. [34] extends grid based Navier-Stokes solvers and level set representations be computed on an unrestricted octree data structure, therefore allowing areas of high detail, such as the interface between the fluid and an object with complex geometry, to be simulated at a higher resolution than other, less interesting areas of the computational domain.

Another important application in computer graphics is interactive applications. Although fluid simulation is reasonably well suited to parallel computational, there has not been any significant work in computer graphics on the parallelization of high quality offline fluid simulation techniques. However, the extension of fluid simulations techniques to the GPU is increasing in popularity as commodity graphics cards provide increased feature-sets and programming tools. An introduction to extending Stam's semilagranian method to the GPU is presented in GPU Gems 2 [26].

There is also a reasonable body of implementation-focused and instructional reference material for fluid simulation methods in computer graphics that is already available and well worth the time of the interested reader [45],[4],[40],[26]. In addition to a qualitative comparison and analysis of fluid simulation methods, this thesis attempts increase this body of work, presenting a solid grounding in the theory of the Navier-Stokes equation and a thorough analysis and comparison of several popular advection algorithms.

II

The Navier Stokes Equation

The Navier-Stokes equation describes the motion of fluid over time. Here we will derive the Navier-Stokes equation as it is seen most commonly in fluid simulation papers in computer graphics. Modern fluid simulation literature in computer graphics is often daunting at first, as the techniques presented are often building on decades of research. Similarly, general fluid mechanics texts are often highly mathematical, and texts aimed at fluid simulation are often geared towards mechanical engineers, focusing on specific forms of the Navier-Stokes equation that often arise in engineering applications, such as modeling airflow over a wing. The goal of this section is to provide the reader with a solid mathematical reference on how the fully general Navier-Stokes equation is derived, with annotations along the way to correct common misconceptions, and provide the mathematical intuition behind physical quantities like *pressure* and *viscosity*.

For simplicity, the derivation is restricted to generalized three dimensional vectors, where applicable, only the x dimension is derived and parts of the derivation that follow trivially for the other dimensions are omitted and left as an exercise. The only prior knowledge for this chapter is an solid understanding of basic calculus, a high-school level physics course, and knowledge of basic vector calculus operations such as gradient, divergence, etc..

The Navier-Stokes equation may be derived directly from Newton's second law of motion:

$$\sum \mathbf{F} = m\mathbf{a} \tag{II.1}$$

However, first we will need to develop a language in which we can talk about the internal forces of fluids. The mechanics of fluids falls into the realm of a more general field known as *continuum mechanics*: the study of materials that are composed of continuous regions of matter. Deriving the properties of a fluid, and indeed many other types of materials, can be accomplished by examining the forces present in an infinitesimally small volumetric region of the material. As we are studying fluids, we will call this region the *fluid volume*; it has dimensions $\Delta x, \Delta y, \Delta z$. To understand the behavior of fluids, we must first reason about the forces acting on our fluid volume, and what they mean. In the language of continuum mechanics, these forces are named *stress* and *strain.*

II.1 Stress and Strain

Stresses and strains on a volumetric element are basic tools used in continuum mechanics, and can be used to derive properties both of solids and fluids. This section gives some of the basic derivations of stress and strain in a fluid volume, and an intuition as how these quantities relate to the physical behavior of a fluid.

A *stress* represents the force acting on the surface of the fluid volume, the stress on this infinitesimal volume can be broken down into stresses acting on each face of our volumetric cube. For the face normal to the x-axis, which we will henceforth call the x-face, there are three stresses:

$$\sigma_x = \lim_{\Delta A_x \to 0} \frac{\Delta F_x}{\Delta A_x}, \qquad \tau_{xy} = \lim_{\Delta A_x \to 0} \frac{\Delta F_y}{\Delta A_x}, \qquad \tau_{xz} = \lim_{\Delta A_x \to 0} \frac{\Delta F_z}{\Delta A_x},$$

Stress represents the force in each direction distributed over the area of the given volume face. The *normal stress*: σ_x (also sometimes denoted τ_{xx}), is the force in the normal direction of the face. The other two are *shear stresses*:

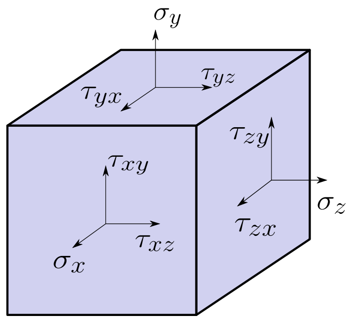

Figure II.1 Orientation of each stress tensor component in 3D, note that the normal stresses are denoted with single subscripts: σ_x which only need one dimension, and shear stresses, labeled with two subscripts, e.g. τ_{xy} indicating both directions of the shear.

τ_{xy} and τ_{xz}, which act in the two directions orthogonal to the normal stress, with the convention that the first subscript denotes the normal direction of the face on which the stress is applied, and the second subscript denotes the direction of the force which induces that stress.

The definition for the shear stresses for the other two normal directions is similar, and illustrated in Figure II.1. This yields 9 scalar quantities to completely describe the stress of any given volume element. One will notice, however, that the x, y and z faces in Figure II.1 give stress values for the top, right, and front faces faces only. The stresses for the bottom, left, and back are defined in terms of the value at the previous faces, *and* their rate of change. For example, for the left face,

$$\sigma'_x = \sigma_x + \frac{\partial \sigma_x}{\partial x}\Delta x \qquad \tau'_{xy} = \tau_{xy} + \frac{\partial \tau_{xy}}{\partial x}\Delta x \qquad \tau'_{xz} = \tau_{xz} + \frac{\partial \tau_{xz}}{\partial x}\Delta x$$

Stresses on the right face are a linear extrapolation of the stresses at the left face, along a distance Δx, and calculations are similar for the y and z faces. This construction for the opposing faces will be used again several times, so it is worth noting.

In addition to stresses on the fluid element, there are also *strains.* Strains

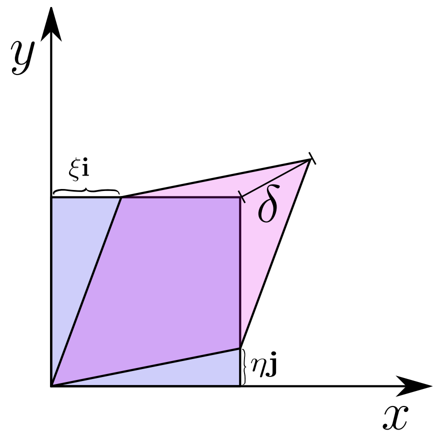

Figure II.2 Strain tensor components, illustrated in two dimensions for clarity: each strain component represents deformation in a given direction, and their sum δ represents the total displacement of upper right corner of the square.

are used to describe an infinitesimal *deformation* of a fluid or solid volume from its initial shape and position. Strains are particularly important to fluids as the deformation of a fluid volume allows one to reason about the change in *volume* of the fluid element. This potential change in volume in turns allows us to reason about the incompressibility of a fluid, and in turn, the concept of *pressure*.

Strains are defined in a very similar convention as stresses. Consider a fluid volume which has undergone a slight deformation: Figure II.2. That is, a vector, $\boldsymbol{\delta} = \xi\mathbf{i} + \eta\mathbf{j} + \zeta\mathbf{k}$ is defined such that if the lower left corner of our fluid is a point O, then the deformed point is $O+\boldsymbol{\delta}$. We can use the same linear extrapolation trick as is used in the construction of our stresses, to define the deformation of the upper right corner $\boldsymbol{\delta'} = (\xi + \frac{\partial\xi}{\partial x}\Delta x)\mathbf{i} + (\eta + \frac{\partial\eta}{\partial y}\Delta y)\mathbf{j} + (\zeta + \frac{\partial\zeta}{\partial z}\Delta z)\mathbf{k}$.

We can derive the elongation of the fluid volume in the x dimension by taking the limit of the difference of each point as Δx goes to zero:

$$\begin{aligned}
\epsilon_x &= \lim_{\Delta x, \to 0} \frac{\left(\xi + \frac{\partial \xi}{\partial x}\Delta x\right) - (\xi)}{\Delta x} \\
\epsilon_x &= \frac{\partial \xi}{\partial x} \\
\boldsymbol{\epsilon} &= \frac{\partial \xi}{\partial x}\mathbf{i} + \frac{\partial \eta}{\partial y}\mathbf{j} + \frac{\partial \zeta}{\partial z}\mathbf{k}
\end{aligned}$$

This gives us $\boldsymbol{\epsilon}$, the *normal strain*, analogous to the normal stress $\boldsymbol{\sigma}$, a deformation in the same direction as the normal of the face produces an elongation or contraction. In addition there are *shear strains* analogous to the shear stresses, defined similarly as the deformations orthogonal to the normal direction of the face. Shear strains produce changes in the fluid volume in the same way as the shear component of an affine transformation matrix; only the orthogonal components of a face are deformed, and parallel lines are preserved.

$$\gamma_{xy} = \frac{\partial \eta}{\partial x} + \frac{\partial \xi}{\partial y}, \qquad \gamma_{yz} = \frac{\partial \zeta}{\partial y} + \frac{\partial \eta}{\partial z} \qquad \gamma_{zx} = \frac{\partial \xi}{\partial z} + \frac{\partial \zeta}{\partial x}$$

Another important characteristic of strains is stress-strain relation, which describes what forces, and subsequently stresses, arise from a deformation of that object. This is the main connection between the shear and stress forces, and is an *intrinsic* property of a material. A piece of wood, for example, will respond differently when deformed, than say a piece of silly putty. For simple elastic solids, the relationship between stress and strain is linear: like a spring, the restorative force is proportional to the distance displaced; this proportionality constant is called the *modulus of elasticity.* Analogous to an elastic solid, a *Newtonian fluid* is a fluid that also has a linear stress-strain relationship, but for a Newtonian fluid, the stress is proportional to the rate of change of the strain; the constant of proportionality, which is analogous to the modulus of elasticity for solids, μ, is known as the *viscosity* of the fluid. Thus the higher the viscosity, the larger the number of internal stress forces are generated within fluid. The fact that the stress-strain relationship depends on the *rate of change of the strain* instead of the strain itself, is intuitive. For example, imagine the difference between raising your hand and slapping a pool of water, and gently easing your hand into the water slowly, and imagine the same actions against a block of wood. This means that highly viscous fluids resist changes in internal velocity while deforming, but not necessarily deformation itself. In the case elastic solids, like the block of wood, the only thing that matters is how much of a deformation has occurred, not how quickly it occurred. This is one of the fundamental distinctions between the behavior of solids and fluids.

If we take the displacement $\boldsymbol{\delta}$ and examine its rate of change with respect to time we really just have the instantaneous velocity of our fluid volume:

$$\frac{\partial \boldsymbol{\delta}}{\partial t} = \left(\frac{\partial \xi}{\partial t}\mathbf{i} + \frac{\partial \eta}{\partial t}\mathbf{j} + \frac{\partial \zeta}{\partial t}\mathbf{k} \right) = (u\mathbf{i} + v\mathbf{j} + w\mathbf{k}) = \mathbf{u}$$

We can now relate the equations for shear stress to the velocity of the fluid via the stress-strain relation; assuming viscosity, μ, is constant over time:

$$\tau_{xy} = \frac{\partial}{\partial t}\left(\mu \left(\frac{\partial \eta}{\partial x} + \frac{\partial \eta}{\partial y} \right) \right) = \mu \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) \tag{II.2}$$

We can do the same for the normal stress, with a bit more work (but we will see, it will be worth it). Let us use the convention thatϵ_x^X is the strain from the normal stress in the x direction. In addition to this, two strains: $\epsilon_x^Y = -n\epsilon_y^Y$ and $\epsilon_x^Z = -n\epsilon_z^Z$ arise from the normal strains in the y and z direction. To what degree this occurs is another intrinsic property of the material, and the ratio n is known as the *Poisson ratio.* The Poisson ratio measures how much the cross section of a block of material will reduce in area as that material is stretched along its axis. The Poisson ratio is usually positive, meaning that there is a negative correlation between the strain in one direction and the strain in its orthogonal directions, i.e. if you stretch an object it becomes longer and skinnier, and if you compress it, it becomes shorter and fatter. The time rate of strain varies linearly with stress, and the Poisson ratio is included in this linear relation as follows [9]:

$$\sigma_x = 2\mu(1+n)\frac{\partial \epsilon_x^X}{\partial t} \qquad \sigma_y = 2\mu(1+n)\frac{\partial \epsilon_y^Y}{\partial t} \qquad \sigma_z = 2\mu(1+n)\frac{\partial \epsilon_z^Z}{\partial t}$$

Next, we will want to find the total strain rate of change in the x direction: $\frac{\partial \epsilon_x}{\partial t}$, so we add the strains in that direction due to all the normal stresses:

$$\begin{aligned}\frac{\partial \epsilon_x}{\partial t} &= \frac{\partial}{\partial t}\left[\epsilon_x^X + \epsilon_x^Y + \epsilon_x^Z\right] \\ &= \mu\left(\sigma_x - \frac{\sigma_y}{n} - \frac{\sigma_z}{n}\right)\end{aligned}$$

By taking the above summation, we can now write expressions for the total strain rate of change in the x direction terms of the x y and z normal stresses:

$$\frac{\partial \epsilon_x}{\partial t} = \frac{1}{2\mu(1+n)}\left[\sigma_x - n(\sigma_y + \sigma_z)\right]$$

The strain rate of change is also the rate of change of position: velocity:

$$\frac{\partial \epsilon_x}{\partial t} = \frac{\partial}{\partial t}\left(\frac{\partial \xi}{\partial x}\right) = \frac{\partial}{\partial x}\left(\frac{\partial \xi}{\partial t}\right) = \frac{\partial u}{\partial x}$$

We now have expressions for velocity components in terms of stresses:

$$\begin{aligned}\frac{\partial u}{\partial x} &= \frac{1}{2\mu(1+n)}\left[\sigma_x - n(\sigma_y + \sigma_z)\right] \\ \frac{\partial v}{\partial y} &= \frac{1}{2\mu(1+n)}\left[\sigma_y - n(\sigma_x + \sigma_z)\right] \\ \frac{\partial w}{\partial z} &= \frac{1}{2\mu(1+n)}\left[\sigma_z - n(\sigma_x + \sigma_y)\right]\end{aligned}$$

We also compute the sum of the rate of change of elongation: $\frac{\partial}{\partial t}(\epsilon_x + \epsilon_y + \epsilon_z)$, as it will be useful later:

$$\begin{aligned}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) &= \frac{1}{2\mu(1+n)}\left(\sigma_x + \sigma_y + \sigma_z - 2n\left(\sigma_x + \sigma_y + \sigma_z\right)\right) \\ &= \frac{1-2n}{2\mu(1+n)}\left(\sigma_x + \sigma_y + \sigma_z\right) \qquad \text{(II.3)}\end{aligned}$$

Rearranging the x velocity in terms of total stresses yields:

$$\sigma_x = 2\mu(1+n)\frac{\partial u}{\partial x} + n\left(\sigma_y + \sigma_z\right)$$

Let us define $\bar{\sigma}$ as the average of all normal stresses as the mean stress: $\bar{\sigma} = \frac{1}{3}(\sigma_x + \sigma_y + \sigma_z)$. We will now do a few algebraic tricks to simplify the expression.

Let us add two terms to both sides of the equation: $-\bar{\sigma}$ and $n\sigma_x$, and then multiply both sides by 3 (don't worry if this seems very unintuitive at first).:

$$\begin{aligned}
\sigma_x + n\sigma_x - \bar{\sigma} &= 2\mu(1+n)\frac{\partial u}{\partial x} + n\left(\sigma_y + \sigma_z\right) + n\sigma_x - \frac{1}{3}\left(\sigma_x + \sigma_y + \sigma_z\right) \\
\sigma_x(1+n) - \bar{\sigma} &= 2\mu(1+n)\frac{\partial u}{\partial x} + n\left(\sigma_x + \sigma_y + \sigma_z\right) - \frac{1}{3}\left(\sigma_x + \sigma_y + \sigma_z\right) \\
3\sigma_x(1+n) - 3\bar{\sigma} &= 3 \cdot 2\mu(1+n)\frac{\partial u}{\partial x} + 3n\left(\sigma_x\sigma_y + \sigma_z\right) - \left(\sigma_x + \sigma_y + \sigma_z\right) \\
3\sigma_x(1+n) - 3\bar{\sigma} &= 3 \cdot 2\mu(1+n)\frac{\partial u}{\partial x} + (1-2n)\left(\sigma_x + \sigma_y + \sigma_z\right) \\
\sigma_x - \bar{\sigma} &= 2\mu\frac{\partial u}{\partial x} + \frac{(1-2n)}{3(n+1)}\left(\sigma_x + \sigma_y + \sigma_z\right) \qquad \text{(II.4)}
\end{aligned}$$

Recall that the change in the sum of elongations we computed in Eqn II.3 also defines the *change in volume* of the fluid, noticing that the left hand side of Eqn II.3 is the divergence of the velocity field.

$$\nabla \cdot \mathbf{u} = \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = \frac{1-2n}{2\mu(1+n)}\left(\sigma_x + \sigma_y + \sigma_z\right) \qquad \text{(II.5)}$$

Substituting Eq. (II.5) into Eq. (II.4), we get:

$$\sigma_x - \bar{\sigma} = 2\mu\frac{\partial u}{\partial x} - \frac{2}{3}\mu\left(\nabla \cdot \mathbf{u}\right)$$

If the change in volume over time is zero, we can say that this fluid is *incompressible*:

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{(II.6)}$$

Recall that we have defined the mean stress as $\bar{\sigma} = \frac{1}{3}(\sigma_x + \sigma_y + \sigma_z)$, and that positive values for stress indicate forces that point away from the fluid volume. Negating the mean stress gives us a quantity that gets larger the higher the sum of forces pushing inwards toward the center of the fluid volume are, in other words *pressure*: $p = -\bar{\sigma}$. We can now write an expression for each of the normal stresses only in terms of pressures and velocities:

$$\begin{aligned}
\sigma_x &= -p + 2\mu\frac{\partial u}{\partial x} - \frac{2}{3}\mu\left(\nabla \cdot \mathbf{u}\right) \\
\sigma_y &= -p + 2\mu\frac{\partial v}{\partial y} - \frac{2}{3}\mu\left(\nabla \cdot \mathbf{u}\right) \qquad \text{(II.7)} \\
\sigma_z &= -p + 2\mu\frac{\partial w}{\partial z} - \frac{2}{3}\mu\left(\nabla \cdot \mathbf{u}\right)
\end{aligned}$$

II.2 Continuous Derivation

We now want to take our equations for each dimension, and come up with a differential equation in vector form. Recalling again Newton's Second law, $\sum F_i = ma_i$, let us write the this equation for the fluid element with density ρ the x direction only.

$$F_x = (\rho \Delta x \Delta y \Delta z) a_x$$

We must now consider the forces acting on the fluid element. As we have seen before, stress is defined as force over a differential area element, and we may reconstruct the force due to stress by summing all the forces in the x direction (note that each face will have exactly one stress component in the x direction). In addition to stress forces, we also want to add an arbitrary *body force*, which is external to the fluid, most commonly gravity, in general we will call this force $\mathbf{f} = f_x\mathbf{i} + f_y\mathbf{j} + f_z\mathbf{k}$.

$$\begin{aligned} F_x = (\rho \Delta x \Delta y \Delta z) f_x - \sigma_x \Delta y \Delta z + (\sigma_x + \frac{\partial \sigma_x}{\partial x}) \Delta y \Delta z \\ - \tau_{yx} \Delta x \Delta z + (\tau_{yx} + \frac{\partial \tau_{yx}}{\partial y}) \Delta x \Delta z \\ - \tau_{zx} \Delta x \Delta y + (\tau_{zx} + \frac{\partial \tau_{zx}}{\partial z}) \Delta x \Delta y \end{aligned}$$

When distributed, each term contains $\Delta x \Delta y \Delta z$, and we can substitute this force back into LHS of Newton's Equation:

$$(\Delta x \Delta y \Delta z)(\rho f_x + \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z}) = (\Delta x \Delta y \Delta z) \rho a_x$$

Our volume element term drops, and we may now substitute values of stress in terms of velocity obtained in Eqs. (II.7) and (II.2) to yield:

$$\rho f_x + \frac{\partial}{\partial x}\left[-p + 2\mu \frac{\partial u}{\partial x} - \frac{2}{3}\mu(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z})\right]$$

$$+\frac{\partial}{\partial y}\left[\mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y})\right] + \frac{\partial}{\partial z}\left[\mu(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x})\right] = \rho a_x$$

Expanding this out yields:

$$\rho f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\mu\left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 v}{\partial y} + \frac{\partial^2 w}{\partial z}\right] + \frac{\mu}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = \rho a_x$$

Similarly for the other dimensions:

$$\rho f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\mu\left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 v}{\partial y} + \frac{\partial^2 w}{\partial z}\right] + \frac{\mu}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = \rho a_x$$
$$\rho f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\mu\left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 v}{\partial y} + \frac{\partial^2 w}{\partial z}\right] + \frac{\mu}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = \rho a_x$$

An important and often confused part of the next step in the derivation is specifying acceleration as the derivative of velocity, specifically the acceleration for a particle of solid mass is $\mathbf{a} = \frac{D\mathbf{u}}{Dt}$, where the operator $\frac{D}{Dt}$ is the *substantial derivative*, also known as the total derivative. The substantial derivative is defined as $\frac{D}{Dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial w}$, the second term, also called the *convective derivative*, can be abbreviated as $(\mathbf{u} \cdot \nabla)$. Confusion often arises, when it is assumed that $(\mathbf{u} \cdot \nabla)$ and $(\nabla \cdot \mathbf{u})$ are equivalent. In fact, they are not even of the same type, the divergence $\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}$ is a scalar, while the convective derivative: $(\mathbf{u} \cdot \nabla) = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial w}$ is a vector operator, where each partial derivative is scaled by the velocity in the direction of that derivative.

The convective derivative is an expression of the fact that the forces acting on a fluid element are dependent on its current velocity, making the differential equation describing fluid flow non-linear. This single term, as we will see, is a large part of why the Navier-Stokes equation is interesting, and what makes the motion of fluids unique compared to other materials. Recalling the form of our equation:

$$\rho f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\mu\left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 v}{\partial y} + \frac{\partial^2 w}{\partial z}\right] + \frac{\mu}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) = \rho a_x$$

If we sum each dimension and convert to vector notation, and replace the sum of accelerations with the substantial derivative, we are left with the *Compressible Navier-Stokes Equation*:

$$\rho\mathbf{f} - \nabla p + \mu\nabla^2\mathbf{u} + \frac{\mu}{3}\nabla(\nabla \cdot \mathbf{u}) = \rho\frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} \qquad \text{(II.8)}$$

As we have seen in Eq. (II.6), applying incompressibility: $(\nabla \cdot \mathbf{u}) = 0$ allows us to drop a few terms, and yields the *Incompressible Navier-Stokes Equation*:

$$\rho \mathbf{f} - \nabla p + \mu \nabla^2 \mathbf{u} + = \rho \frac{\partial \mathbf{u}}{\partial t} + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} \tag{II.9}$$

Dividing by ρ and rearranging the terms, we can alter the equation to appear as it is commonly cited in CFD literature in computer graphics:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u} \tag{II.10}$$

Note that we have made the substitution $\nu = \frac{\mu}{\rho}$. This is commonly done in CFD literature, where μ is termed the *dynamic viscosity* and ν, termed the *kinematic viscosity* is simply the dynamic viscosity normalized with respect to density.

II.3 Overview

Now that we have derived the full incompressible Navier-Stokes equation, let us break it down term by term. We will see that due to the *separability* of differential equations, we can solve each term independently and apply these operations in serial, therefore it is useful to obtain and intuitive mathematical and physical understanding of each term and any additional constraints, such as incompressibility: Recall Eqs. (II.10, II.6) which comprise the incompressible Navier-Stokes equation:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u}$$
$$\nabla \cdot \mathbf{u} = 0$$

Advection: $-(\mathbf{u} \cdot \nabla)\mathbf{u}$. This is likely the most complicated term, as we will see when we discuss methods of solution, and in fact it is what makes the Navier-Stokes equation a non-linear differential equation since it contains two factors of $\mathbf{u}$. Advection is the transport of any quantity through a vector field. For example, particles moving through a vector field are being *advected*, scalar quantities such as temperature may also be advected through a vector field. Advection may also be conducted on vector as well as scalar fields. Even further, a vector field can advect itself. This is called self-advection, and is exactly what the advection term in the Navier-Stokes equation describes. It is difficult to provide further intuition, other than to say that this forms the basis of how to establish a rule to advance from one timestep to the next, as we are modeling future velocity as some function of the current velocity.

Diffusion/Viscosity: $\nabla^2\mathbf{u}$, is responsible for simulating the viscosity of a fluid. Intuitively this type diffusion can be thought of as iteratively applying a smoothing filter. Velocities close together will tend towards the same value. The higher the viscosity, the faster the velocities will diffuse. This explains why viscous fluids come to rest much sooner once disturbed, and have less 'lively' behavior.

Pressure: $-\frac{1}{\rho}\nabla p$ accounts for the velocity due to differences in pressure of the fluid. Consider two adjacent fluid volumes: if the pressure is very high in one fluid volume, and low in all the adjacent fluid volume, fluid will want to move out to these low pressure areas (this is why it is the negation of the divergence field, which points in the direction of highest pressure increase).

Body Force: $\mathbf{f}$, last term is the free body force which represents any external force applied to the fluid. This is often gravity, but it need not be constant; it can be an arbitrary function over space, time, a parameter of the simulation (such as temperature or density), or a forcing function used for artistic control.

Continuity/Incompressibility: $(\nabla \cdot \mathbf{u}) = 0$. While not part of Eq. (II.10), in order to simulate an incompressible fluid, this condition must be enforced. It states that the divergence of the velocity field is zero everywhere. This means that for any fluid volume, the amount of fluid coming in and the amount of fluid going out is conserved. This equation is exactly what enforces the incompressibility constraint. Empirically this is what causes a vector field to be "swirly", when forces are applied inside the fluid; the reason for this is that when a force is locally applied to a uniform field, it inherently causes divergence. Since the field must be divergence free everywhere, it is often that vortex patterns emerge as they are the only way large velocities can exist and still maintain a locally divergence free field; each fluid volume in a vortex has an equal amount of fluid coming in as it does going out.

III

Grid Based Methods

As noted in the first chapter, we will only concern ourselves with grid based fluid simulation methods, but first we must define what constitutes a 'grid based' method. The concept of a regular computational grid arises intuitively from the discretization of the variables in the differential form of the Navier-Stokes equation, as well as the infinitesimal volume element analogy that we have used to derive many of our continuous expressions. As we are dealing with properties of a fluid that vary over space and time, we must break each of these dimensions into discrete 'chunks'.

Discretizing time, is straightforward as it is one dimensional. Discretizing multi-dimensional space, however, is somewhat more complex, as there are a variety of ways to partition two and three-dimensional space. In fact, complex tetrahedral and polygonal mesh discretizations are commonly used in the engineering and aerospace communities, as they are able to handle fluid/non-fluid interfaces with complex geometries more accurately [1]. Generally in computer graphics applications, the accuracy at the fluid interface gained from these mesh based discretizations is not worth the added complexity, and simple regular discretizations are favored.

A 'grid-based' discretization is one in which all of the discrete elements are 'boxes' (i.e. cuboids). This grid is most commonly formed by taking an existing volume, usually the entire domain in which we are interested in simulating our fluid, and dividing each dimension into a discrete number of chunks. The number of these divisions define the resolution of the grid, and subsequently the size and shape of each grid cell. For the purposes of this discussion, we will also only consider *uniform* grids, where the width and the height of a grid cell are fixed over the entire domain. This is in contrast to what are commonly referred to as *adaptive* grids, whose cell sizes are spatially varying. In addition to this constraint, to simplify the discussion, we will assume that the size of grid cells are constant equal in all dimensions, i.e. $\Delta x = \Delta y = \Delta z$.

III.1 Grid Cells & Variables

As we have variables that vary spatially, and we have now split up our space into discrete chunks, we must define a way to assign these variables to each subdivided chunk, essentially we are sampling continuous quantities like velocity, pressure, density, viscosity, etc. at each grid cell. An intuitive approach is to simply take the center of each grid cell, and define all of our quantities there. This approach has several advantages, it's easy to understand and it works well with certain algorithms [44], but this need not be the only choice.

A common choice historically in CFD literature for how to define variables on a grid is what's know as a *staggered grid*, this is in contrast to the cell-centered approach above, known as a *collocated grid*. To motivate the intuition behind a staggered grid, think about how one would place variables on a one dimensional line that had divided up into many segments. Would it be at the center, or at boundary between each division? The intuition is not as straightforward here, and requires an evaluation of what exactly the quantity is trying to represent to decide which location is appropriate. In a staggered grid, some quantities are defined at the center of the grid, most notably pressure, while others, such as the velocities, are defined at the divisions between cells; in two dimensions, these are the lines adjoining cells, in three dimensions these are the cell faces.

It is also useful to assign names or indices to these cells. Usually this done by setting an arbitrary location in the computational domain (usually one of the corners) as index $(0, 0)$ with aligned axes defining cell increments in each respective dimension. For this discussion we will use the convention that the $(0, 0)$ cell starts in the 'bottom-left', the positive x, and y directions are oriented to the right and up, with respect to the page.

We can uses these indices to establish some conventions about how we name the positions on our grid. For any continuous field variable q we can denote $q_{i,j}$ as the value of q at the center of cell (i, j). This gives us a way to precisely name any location in our grid, including the boundaries between adjacent cells. For example, for the x-direction velocity stored at the face adjoining the origin cell: $(0, 0)$ and the cell to its right $(1, 0)$ we denote as $u_{i+\frac{1}{2},j}$, similarly for all the other faces.

The trickiest part of a staggered grid configuration is that some quantities, such as velocity are stored natively at the cell faces, and must be interpolated if we want to obtain the velocity at the centers of the cell, whereas values such as pressure are defined at the cell centers and must be interpolated to find the value at the cell faces.

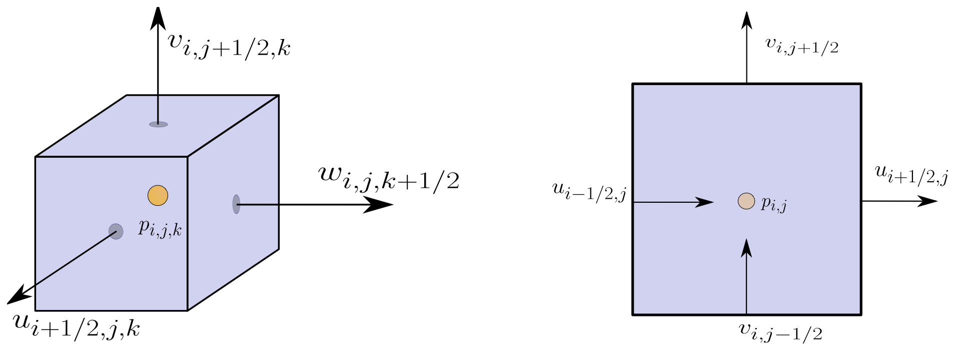

Figure III.1 Velocity and pressure on a staggered grid in two and three dimensions respectively. In three dimensions, the velocities on the opposing faces, while still present, are omitted for clarity.

III.2 Marking the Fluid

One very important part of simulating fluids, particularly liquids, is marking where in space the fluid is present. This is one problem that faces any grid based method: generally for a computational grid to be efficient in calculating velocities, it must be fairly coarse. As a result, the method used for representing the actual geometry of the fluid is usually a totally different, and often higher resolution, representation of the geometry than the grid cells that carry and compute the velocity and pressure. This combination has yielded the term *Marker and Cell* method, often abbreviated the *MAC* method.

In early developments in computer graphics, the preferred method of marking the surface was a heightfield [28]. Heightfields are simple and compact representations, but are single valued in height. In later fluid simulation research, heightfields were abandoned in favor of a collection of particles, or an implicit surface, usually a level set or a signed distance function. The idea behind the MAC method is that at the beginning of every time step, the fluid markers, whatever they be, are used to provide information that the solver needs to compute the velocity on the grid cells. The most important piece of information that the fluid markers provide to the solver is which grid cells are full, and which are empty, and as a corollary, which cells have faces open to air and which are fully surrounded by fluid (this is important when determining pressure forces acting on the fluid). Additionally, some marking methods can provide extra information about curvature of the surface for computing surface tension, or act as additional locations to sample fluid velocity.

At the end of the time step, the fluid markers are advected by the velocity field to get their updated position and shape for the next timestep. The method by which the markers are advected through the velocity field is entirely dependent on the marker representation. Any geometric representation can be used as a basic fluid marker as long as the grid cells can query it to determine whether they contain fluid, and the marker can be advected by the velocity field. In this manner, the marker and grid cell calculations are coupled. This is significant, because it is often assumed or implied that the fluid markers are 'totally passive' with respect to the solver calculation. For liquids, this is far from the truth. A very important part of simulating liquids is marking the *free surface*, or the interface between fluid and non-fluid.

For fluids, this interface is the only aspect of the fluid simulation that viewers will actually see, and if there is an error or inaccuracy in marking the free surface, a correct velocity solver will still produce incorrect or inaccurate results. When simulating gasses and smoke, markers are still used to determine concentrations of scalar quantities in present, (such as temperature, or smoke density), but the entire computational domain can be considered fluid, so the coupling of markers and grid cells is only one-way.

All of the fluid simulations implementations discussed here use collections of particles exclusively as a marking method for free surface of liquids, and scalar density fields for concentrations in gasses. The primary reason for this is simplicity. Level set representations are very powerful and can represent fluid-air interfaces at high resolutions, but carry additional burdens of computational efficiency and infrastructure, and the best method to advect level sets is the subject of considerable ongoing research. [13],[14] [33], Additionally, some solvers tested, such as PIC/FLIP require particles for their method of solution; It is worthwhile to note that particles can be converted to smooth level sets as a post-process when high quality 3D rendering is required [53].

III.3 Discretizing Navier-Stokes

Now that we have defined the location and types of variables on our staggered grid, we can now take the Navier-Stokes equation and discretize each term. This discretization will form the basis for how we solve for each of these terms. For simplicity we will only consider a two dimensional discretization, and

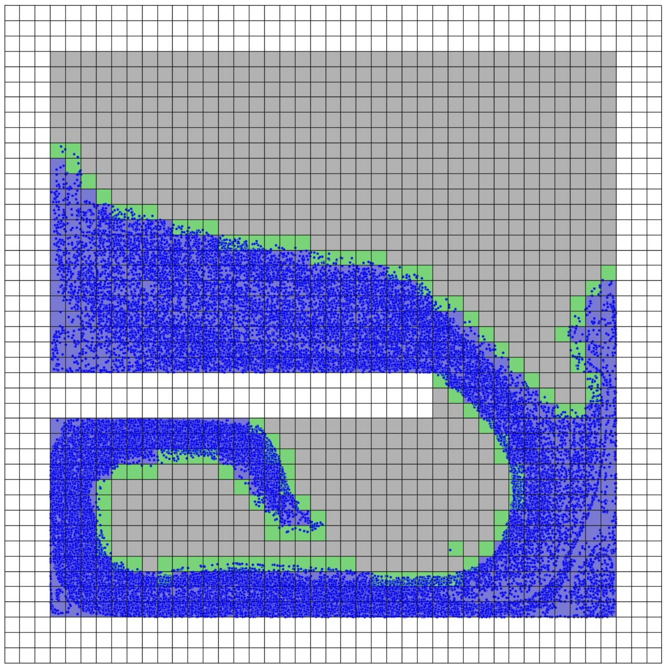

Figure III.2 A visualized example of a marker-and-cell fluid simulator in which full fluid cells are colored blue, fluid cells adjacent to empty cells are colored green, and empty cells are gray. Particles marking the fluid are also shown.

note cases in which an extension to three-dimensions does not follow trivially: Recall Eq. (II.10):

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}$$

Let us write this as the change in velocity for each direction:

$$\frac{\partial u}{\partial t} = f_x - \frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} + \right] + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}$$
$$\frac{\partial v}{\partial t} = f_y - \frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left[\frac{\partial^2 v}{\partial x} + \frac{\partial^2 v}{\partial y} + \right] + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}$$

One important thing to note at this point is that the Navier-Stokes is a separable partial differential equation. Separability means that each term of the equation is independent from the others, and can be solved for independently. This allows us a great advantage, as the numerical method that is best suited to each term can be used. Here we will present and discretize each separable term as a separate equation.

III.3.A The Pressure-Poisson Equation

The pressure term can be simply discretized as a first order difference:

$$\frac{\partial u}{\partial t} = \frac{\partial p}{\partial x} = \frac{p_{i,j} - p_{i+1,j}}{\Delta x}$$

Ordinarily, this might create problems because this appears to be a forward difference, and in order to get a symmetric centered difference, we would have to widen difference stencil to $p_{i-1,j} + 2p_{i,j} + p_{i+1,j}$. The result of the difference between two pressures is a velocity (in the same way that the gradient of a scalar field is a vector field), so the result we want from differencing the pressure at two adjacent cells is the velocity at the *face* of the cell. This is where the elegance of the staggered grid configuration shines: first order differences on a cell-centered quantity become centered differences for the face-centered quantity, and vice versa. Similarly, if we want to compute the pressure due to a difference in velocity, a difference of: $u_{i-\frac{1}{2},j} - u_{i+\frac{1}{2},j}$ will be centered around the cell-center, where the pressure resides.

This is fortuitous, as the general method of solution of pressure takes the form of first computing the induced pressures from the velocity grid, generating a new set of pressures that will enforce the incompressibility of the fluid, and then applying these pressures to get a set of velocities. This process is often referred to as 'pressure projection'. This subject has been covered well in a variety of sources [4], [24], [44], [45], [8], so we will present only a brief overview.

Now that we know how to get pressures induced by velocity and velocity induced by pressures, we must now figure out a way to actually compute the pressure values that lead to an incompressible fluid. To motivate this, let us consider our incompressibility equation: $\nabla \cdot \mathbf{u} = 0$, which says the divergence of the velocity field is zero everywhere. When discretized, this means that on a single grid cell, the face-velocities sum to zero: $u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j} + v_{i,j+\frac{1}{2},} - v_{i,j-\frac{1}{2}} = 0$. Intuitively if mass is being advected through our fluid volume, the same amount of mass coming in is the same amount that leaves, i.e. there is no compression or expansion occurring. All we need to do is solve for this pressure field, and then apply the velocities induced by those pressures to obtain our incompressible and divergence free field. Using previously derived results [44], [8], we can show that the pressure field p can be be calculated using the equation:

$$\nabla \cdot u = \nabla^2 p$$

This form of differential equation is known as *Poisson's equation.* On the right hand side we are solving for the pressure, p represented as a scalar field with values in the cell centers. The left hand side, we have the divergence of the field before pressure projection, which we can compute and treat as a constant. This relationship can be formulated as a matrix and solved using a linear system.

An excellent treatment of how to construct and solve this linear system is presented by Mark Carlson's Phd. thesis [4]. Carlson uses a conjugate-gradient linear solver to solve this matrix, but it is important to point out that this is not the only method available. Harlow [24] and Stam [45] both use relaxation methods to solve the pressure equation. It is important to note that while these methods may look very different in implementation, they are still constructing and solving the same linear system.

III.3.B The Diffusion Equation

For the diffusion term:

$$\frac{\partial u}{\partial t} = \nu \left[\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} \right]$$

Let us examine the x partial derivative:

$$\begin{aligned}\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x} &= \nu \frac{\partial}{\partial x} \left(\frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{\Delta x} \right) \\ &= \nu \left(\frac{u_{i+1,j} - u_{i,j}}{\Delta x} - \frac{u_{i,j} - u_{i-1,j}}{\Delta x} \right) / \Delta x \\ &= \nu \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}\end{aligned}$$

The diffusion equation can be solved exactly using the finite difference scheme above, and this works well in practice for low viscosity flows, however when the viscosity becomes moderately high, this differencing scheme becomes unstable. Like the Pressure-Poisson equation above, the general diffusion equation: $\frac{\partial u}{\partial t} = \nu \nabla^2 u$ is a commonly recurring problem in physical science, and a large number of implicit methods exist for its solution. One will notice a resemblance to *Poission's equation* above, as a result, this equation can be solved with similar methods [4] [5].

III.3.C Advection

One of the most interesting discretization is the advection term:

$$\frac{\partial u}{\partial t} = -\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right)$$

This equation can be solved in a wide variety of ways, and the method of solution here will have the largest impact on the behavior of our fluid solver, and is the primary difference in technique discussed in papers. As such, we describe four advection methods in detail.

First, it is useful to point out that the above equation is the special case of advection: *self-advection*: $(u \cdot \nabla)u$. In general, the advection operator, or convective derivative, $u \cdot \nabla$ can be applied to any scalar or vector field, not necessarily a velocity field, so we can write the general advection equation on a scalar field ϕ as:

$$\frac{\partial \phi}{\partial t} = -\left(u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y}\right)$$

One can substitute u for ϕ in the discussion, but we will use ϕ, as it serves to illustrate that the fact that u or v are both the advect*ing* and advect*ed* fields is purely coincidental for the purposes of solving the equation. They could be temperature, or the density of a chemical solution suspended in the fluid, and the solution would be the same.

First Order Differencing

The most basic method of solving the advection equation is to use explicit finite differencing on the advection equation.

We can define the partial derivative using a first order centered difference, e.g.

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x}$$

$$\begin{aligned}\frac{\partial \phi}{\partial t} &= -\left(u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y}\right)\\ &= -\left(u\left(\frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x}\right) + v\left(\frac{\phi_{i,j+1} - \phi_{i,j-1}}{\Delta y}\right)\right)\end{aligned}$$

In order to get an actual equation for $\frac{\partial u}{\partial t}$ and $\frac{\partial v}{\partial t}$, we can replace ϕ above with u and v respectively.

$$\begin{aligned}\frac{\partial u}{\partial t} &= -\left(u_{i,j}\left(\frac{u_{i+1,j} - u_{i-1,j}}{\Delta x}\right) + v_{i,j}\left(\frac{u_{i,j+1} - u_{i,j-1}}{\Delta y}\right)\right)\\ \frac{\partial v}{\partial t} &= -\left(u_{i,j}\left(\frac{v_{i+1,j} - v_{i-1,j}}{\Delta x}\right) + v_{i,j}\left(\frac{v_{i,j+1} - v_{i,j-1}}{\Delta y}\right)\right)\end{aligned}$$

We do have to be mindful, however, if we are using a staggered grid. For a staggered grid we must make the following approximations:

$$\begin{aligned}u_{i,j} &= \frac{u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}}{2}\\ v_{i,j} &= \frac{v_{i,j+\frac{1}{2}} + u_{i,j-\frac{1}{2}}}{2}\end{aligned}$$

However, several terms cancel, and this substitution results only in a factor of 2 division; if we define a simplified notation for our staggered grid as

$u^s_{i,j} = u_{i-\frac{1}{2},j}$:

$$\begin{aligned} u_{i+1,j} - u_{i-1,j} &= (\frac{u_{i+\frac{3}{2},j} + u_{i-\frac{1}{2},j}}{2}) - (\frac{u_{i-\frac{1}{2},j} + u_{i-\frac{3}{2},j}}{2}) \\ &= \frac{u_{i+\frac{3}{2},j} - u_{i-\frac{3}{2},j}}{2} \\ &= \frac{u^s_{i+1,j} - u^s_{i-1,j}}{2} \end{aligned}$$

QUICK

The QUICK method was first introduced into the CFD literature by Leonard [31] for use in ocean models, and has subsequently been modified and improved by several ocean modeling researchers [47] [51]. One reason for the detail in our first-order differencing derivation above, is that the QUICK advection scheme is easy to understand in the context of the original derivation.

In the first-order differencing method above, we defined the partial derivative using a first order centered difference, e.g.

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x}$$

The crux of the QUICK method is that we instead use a second order approximation:

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x} - \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{8\Delta x}$$

This can be shown to produce a second order accurate solution, with error of $O(\delta x^2)$ instead of $O(\delta x)$ as in the standard first-order differencing algorithm. Another method, dubbed *Modified Split-Quick*, is fourth-order accurate, and is obtained using the approximation:

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x} - \frac{\phi_{i+2,j} - 2\phi_{i+1,j} + 2\phi_{i-1,j} - \phi_{i-2,j}}{12\Delta x}$$

We use the Modified Split-Quick method, which provides a more accurate solution at very little additional computational cost. Details and an error analysis of both of these methods can be found in Webb et al. [51].

Semi-Lagrangian

The semi-lagrangian advection scheme is likely the most commonly used in all of computer graphics, largely due to the paper "Stable Fluids", presented by Jos Stam at SIGGRAPH '99 [44], in which this method is discussed in detail. It is the intention of this section to give a brief overview of and intuition behind the semi-lagrangian method that is left out of the original paper, and a reader desiring a more thorough treatment should consult the above reference.

Although this method was already well known in CFD literature at the time [8], Stam provided a detailed and concise explanation, as well as applications and extensions for simulating smoke and gas, and substantial reference material for implementation [45]. The main advantage of the semi-lagrangian method is that it is *unconditionally stable*, that is, the simulation is guaranteed not to diverge, unlike many finite differencing schemes (which includes all of the ones discussed in this paper). This guarantee of stability and Stam's thorough treatment of this method caused it to become the de-facto standard advection algorithm in computer graphics.

In contrast to finite differences, which can be considered a fully 'eulerian', or fixed reference frame method in which the velocity at a fixed point in space is computed based on its surrounding velocities, lagrangian methods are those which utilize a moving frame of reference, such as along the path of a particle that is being advected. Because of this, lagrangian methods are excellent for computing advection. In particular, if we have a particle sampling the field ϕ with velocity (u, v), the advection equation:

$$\frac{\partial \phi}{\partial t} = -\left(u\frac{\partial \phi}{\partial x} + v\frac{\partial \phi}{\partial y} \right)$$

can be trivially solved by simply advancing the particle forward by its instantaneous velocity sampled at the particle location itself.

After our particle has been advected, we've essentially "pushed" this sample forward in the direction of velocity, which is the essence of the advection equation. The problem is that taking this result and propagating it back to the grid can be tricky. Depending on the velocity, the particle might end up anywhere in our fluid domain; and the grid only stores values at cell faces.

The solution that the *semi*-lagrangian method takes is instead to advect the particle *backward* in time, determining the value $\phi_{i,j}^{old}$ at an arbitrary location on the grid, and interpolating between neighboring grid cells to get the result. We then replace $\phi_{i,j}$ with $\phi_{i,j}^{old}$, the reasoning being that at the next timestep, a particle positioned at $\phi_{i,j}^{old}$, will be advected and arrive at position of $\phi_{i,j}$. This makes the semi-lagrangian method an *implicit* solution to the advection equation.

This backwards extrapolation also gives the semi-lagrangian method unconditional stability; the solution can never diverge as the magnitude of the advected velocity can be no greater than the velocity at some other location that already exists in the fluid. This method also has the unfortunate side effect of smoothing the velocity field, as the both the steps of advecting the particle back in time and and determining its velocity at its advected position require interpolating the grid velocity at arbitrary points in space. This drawback can be mitigated by using higher order interpolation [16], but this is costly, and is by no means completely removed.

PIC and FLIP

The PIC or *Particle in Cell* method was developed by Harlow in 1964 [25], and later extended to the FLIP, or *Fluid Implicit Particle* method by Brackbull et al. in 1986 [27]. It was introduced to the graphics community by Zhu et al. for use in simulating sand as a fluid, but with broader applications as well [53] [52]. All of the above references are excellent sources for these methods described in additional detail.

Like the semi-lagrangian method described above, the PIC and FLIP methods are hybrid eulerian-lagrangian methods. These methods differ in how they attempt to resolve the difficulty of transferring the result of lagrangian particle advection to an eulerian grid. Where the semi-lagrangian method recasts the problem into an implicit framework, the PIC and FLIP methods retain explicit forward integration of particles, but then, using a large number of particles to densely sample the velocity, transfer the particle velocities back to the grid after they are advected.

In both of these methods, all of the other Navier-Stokes calculations such as calculating viscosity, pressure, body forces, etc. are done using standard grid-based methods. At the advection step, however, the velocity at the grid cells is transferred, using interpolation, to the particles. The particles are then advected with their instantaneous velocity, and then their velocity is transferred back to the grid cells.

This process is exactly how the PIC method operates, but one will notice that this involves a lot of interpolation, which results in damped out and smoothed velocity fields, just as in the semi-lagrangian case. The FLIP method, instead, takes a much greater advantage of the particles as additional places to sample the velocity. In the FLIP method, at the beginning of the timestep, the velocities are transferred from the particles to the grid, and this velocity is saved: $\mathbf{u}'$. All of the standard grid-based operations such as the diffusion and pressure equations are computed on the grid and a new velocity is saved as $\mathbf{u}$. Now instead of overwriting the velocity of the particles with the interpolated velocities at $\mathbf{u}$, the difference of the two velocity fields: $\mathbf{u} - \mathbf{u}'$ is interpolated and added to each particle. The result is that only the *change* in velocity produced by the grid based methods is transferred to the particles.

One unfortunate side-effect of the FLIP method that has not been addressed in literature is that changes in grid topology can often have a significant influence on the velocity that is added to the particles. This is illustrated in Figure III.3; due to the discrete nature of the grid and the hard requirement of zero divergence and incompressibility, velocities can change in large magnitudes that are independent of timestep. As a result, if timesteps are small, this can produce a near-divergent velocity addition to the particles and cause them to move in wild directions. While the method still remains unconditionally stable, these artifacts are noticeable and can be quite prominent when small timesteps are used.

It is worth noting that this behavior was independently observed both in our implementation and a reference implementation provided by Zhu & Bridson. In practice, as suggested by Bridson, this behavior can be reduced by blending the result of the FLIP calculation with a small percentage the PIC calculation. For example 99% of the particle velocity is determined by the existing particles velocity and the difference field $\mathbf{u} - \mathbf{u}'$, and 1% of the velocity is determined by interpolating $\mathbf{u}$ directly, however this adds unwanted viscosity to the entire solution. An alternative to this method is to utilize the stabilization technique for explicit methods discussed in Chapter IV.

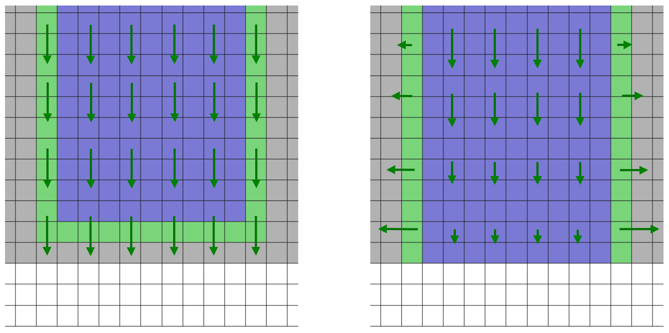

Figure III.3 Change in velocities due to instantaneous changes in pressure, the figure illustrates the velocity field just before the collision (left) and just after (right), due to the nature of the grid based marker cells, this transition can change in a single arbitrarily small timestep.

III.3.D External Forces

The last term in the equation is $\mathbf{f}$, the body force. This term requires no solution per se, but it plays a significant role in the value of the velocity grid when marker cells are completely, or nearly completely surrounded by non-fluid grid cells.

In Harlow's original treatment of marker-and-cell techniques, he stated that when a cell is surrounded on the top and bottom or left and right by empty cells, the velocities should be computed using the body force only [24]. This condition works reasonably well in simple circumstances, but he did not specify how velocities should be assigned when cells are adjacent to solid boundaries.

The reason this creates problems is that if we enforce the divergence free condition on cells adjacent to solid boundaries, we can run into situations where a cell becomes 'stuck' against a solid boundary cell, whose velocities are constrained to be zero. This can also cause problems with explicit finite difference solvers, which can quickly diverge when these stuck cells are within the range of the difference stencil.

This issue is often neglected in literature, but has been observed and dealt with in a few sources [3] [40]. To prevent stuck cells, we take a different approach that leverages the constraint of only using particles to mark our surface. While advecting our marker particles, we check the cell the particle is in for a special *free-fall* condition. The free-fall condition is true when the following conditions are met:

- the cell is a *surface cell* (it adjacent to at least one empty cell)

and one of the following is true:

- the cell is not adjacent to a *interior cell* (a fluid cell that is not a surface cell)
- the cell is adjacent to both an empty cell and two or fewer boundary cells

If this free fall condition is met, then we ignore the forces defined on the computational grid, and instead move the particle forward by the body force $\mathbf{f}$ only. In practice, this prevents the stuck cell problem and maintains realistic behavior.

III.4 Leapfrog Integration

One aspect of developing a fluid solver, and indeed any physical simulation is the question of what method of *integration* is to be used. Numerical integration is an entire body of academics itself, so we will not attempt be exhaustive. Integration is what allows us to actually advance from one timestep to the next, for example we have an equation of the form:

$$\frac{\partial u}{\partial t} = f(u)$$

In our case $f(u)$ is the Navier-Stokes equation. If we want to discretize this equation and solve it, a simple solution would be analogous to what we did for our Finite Difference method:

$$\begin{aligned} \frac{\partial u}{\partial t} &= f(u) \\ \frac{u_{t+\Delta t} - u_t}{\Delta t} &= f(u_t) \\ u_{t+\Delta t} &= u_t + \Delta t \cdot f(u_t) \end{aligned}$$

This is the simplest and most basic type of integration knows an *forward-euler* integration. Essentially we are taking the instantaneous derivative of u at time t and linearly extrapolating the function's value a distance of Δt value based on that derivative. This is obviously not accurate when Δt is large, but as $\lim_{\Delta t \to 0}$, we approach the accurate solution. Similarly we can define *backward-euler* as:

$$\begin{aligned} \frac{\partial u}{\partial t} &= f(u) \\ \frac{u_t - u_{t-\Delta t}}{\Delta t} &= f(u_t) \\ u_t &= u_{t-\Delta t} + \Delta t \cdot f(u_t) \end{aligned}$$

Unfortunately we must live in the world of realistic timesteps, so we must improve the properties of our integration methods without increasing the timestep. There are many others: backwards integration, similar to what we use for the semi-lagrangian, Runge-Kutta integration, etc.. One method that works particularly well for fluid simulation is *Leapfrog* integration [51] [2].

The basic idea behind Leapfrog integration is very similar to staggering the spatial grid, except we are staggering temporal samples instead of spatial ones. We can derive Leapfrog integration by combining both forward and backward euler:

$$\begin{aligned} u_{t+\Delta t} &= (u_t) + \Delta t \cdot f(u_t) \\ u_{t+\Delta t} &= (u_{t-\Delta t} + \Delta t \cdot f(u_t)) + \Delta t \cdot f(u_t) \\ u_{t+\Delta t} &= u_{t-\Delta t} + 2\Delta t \cdot f(u_t) \end{aligned}$$

Leapfrog integration is particularly well suited to discretized equations whose solutions a undergo periodic oscillation. In these cases, the leapfrog method prevents these oscillations from causing the solution to diverge [2].

One Dimensional Channel Advection

In order to get a qualitative measure of the effect of different techniques, such as integration methods, have on the solution of the Navier-Stokes equation, we present a one-dimensional calculation of the advection term on a simple scalar profile. This technique is commonly used in CFD literature to as a tool to compare the accuracy of a method used to solve the advection problem (cite several papers that use this technique).

For example, Figure III.4 illustrates the difference between using explicit forward euler advection and leapfrog advection using first order central differences. Observing the two contrasting plots, it is evident that the advection solution is unconditionally unstable under forward euler, while it is conditionally stable (albeit noisy) under the leapfrog integration method.

III.5 Flux Limiting

Flux limiting is a technique that mixes two methods to solve the Navier-Stokes equation: one method is non-dissipative but need not be stable, and one

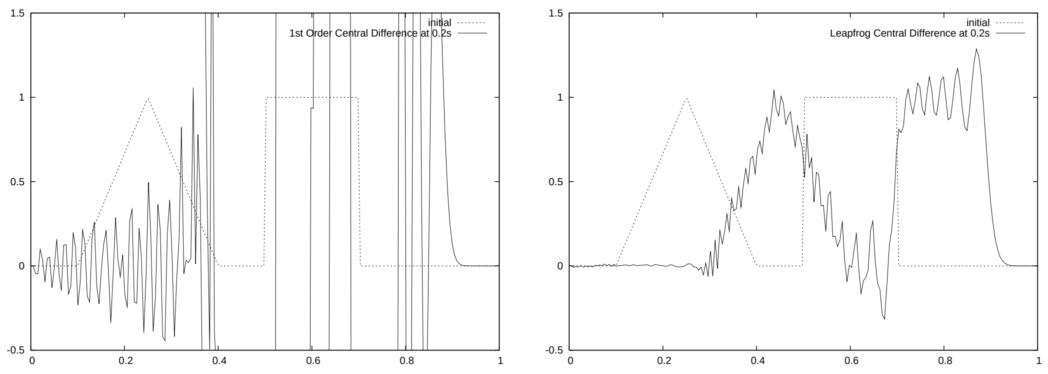

Figure III.4 Comparison of first-order central differences explicit forward euler integration (left) and leapfrog integration (right).

method must be stable but may also be dissipative (as these two properties tend to be correlated).

The idea is that when one encounters a "shock", or a rapid change in space in the velocity field; often caused by fluid encountering a boundary the stable method is used. When the velocity is changing slowly, the unstable but non-dissipative method is used. The decision of how much each of each method to use, and when, is defined by the *flux limiting function*, many of which have been developed at different times and for different applications.

The first step in flux limiting is to compute the ratio of adjacent velocities, so for each grid cell velocity, we compute:

$$r_u = \frac{u_{i,j} - u_{i-1,j}}{u_{i+1,j} - u_{i,j}}$$

$$r_v = \frac{v_{i,j} - v_{i,j-1}}{v_{i,j+1} - v_{i,j}}$$

This value, in the range $[-\inf, +\inf]$, gives a metric of how quickly the velocity is changing, and thus the magnitude of the 'shock'. r_u and r_v are the input to a flux limiting function: $\Phi(r)$.

The output of $\Phi(r)$ determines how much of the stable or the unstable solution to use. Generally the flux limiting operation is as follows.

We use a variation on the Van Albada flux limiting function [29]:

$$\Phi(r) = \frac{2r}{r^2 + 1}$$

This flux limiting function yields 0 when the measure of shock is large, and 1 when the measure of shock is low. Therefore we can implement our flux limiting as follows:

$$u_{i,j} = lerp(\Phi(r_u), u_{i,j}^{stable}, u_{i,j}^{unstable})$$

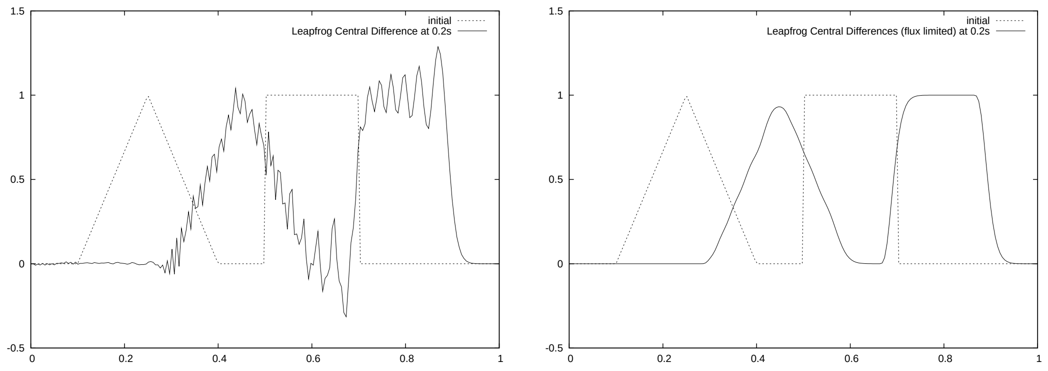

Figure III.5 Comparison of the first-order central differences using leapfrog integration without (left) and with (right) the flux limiter.

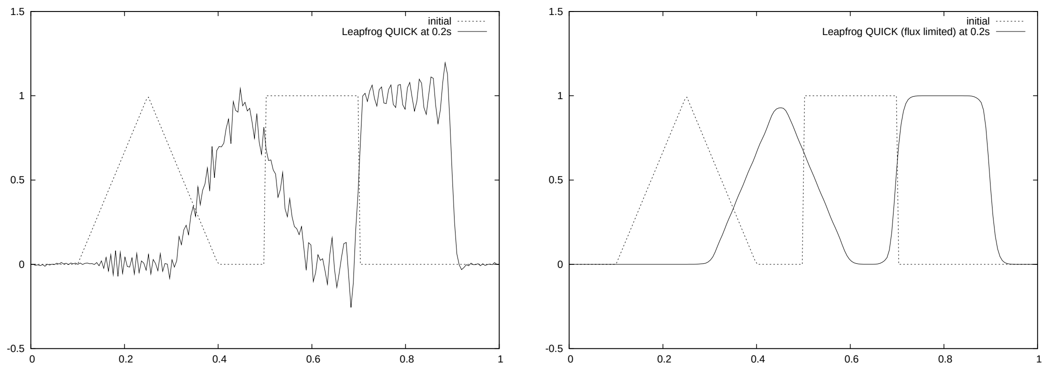

Figure III.6 Comparison of the QUICK advection using leapfrog integration without (left) and with (right) the flux limiter.

III.6 Comparison of Solvers

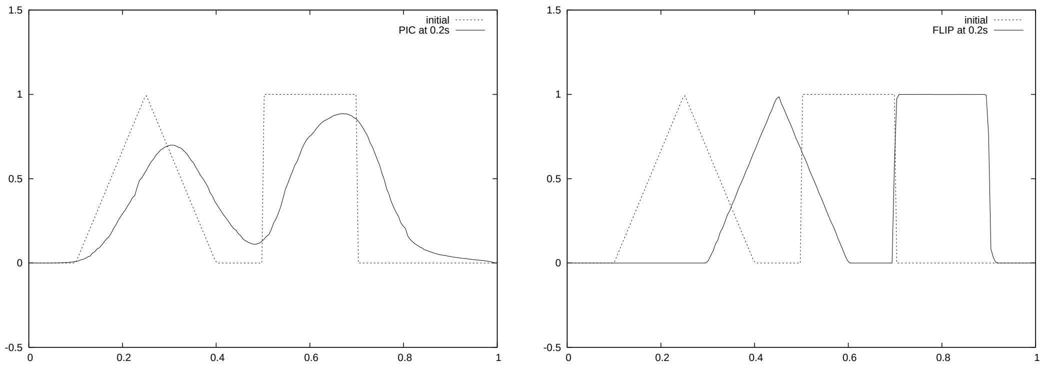

Figure III.7: Comparison of the PIC (left) and FLIP (right) particle methods.

We compare the PIC vs. FLIP methods in Figure III.7. For this comparison, particles were initialized in our one-dimensional grid, with four particles randomly positioned per grid cell (similar to multi-jittered pixel sampling). It is evident from the plot the degree to which PIC method introduces artificial viscosity. The FLIP method, on the other hand, creates a near perfect reconstruction of the advection profile, with no noticeable dissipation, since the particles are simply being moved, and no data is lost or interpolated at any stage. This is a slightly contrived example, as the pressure and boundary condition forces significantly alter the behavior of the FLIP method, but it correctly characterizes this advection method as having little to no intrinsic dissipative effects or artifacts.

In figure III.8 we compare the results of each of the four primary methods we have discussed. These methods are compared using two-dimensional solutions in the next chapter.

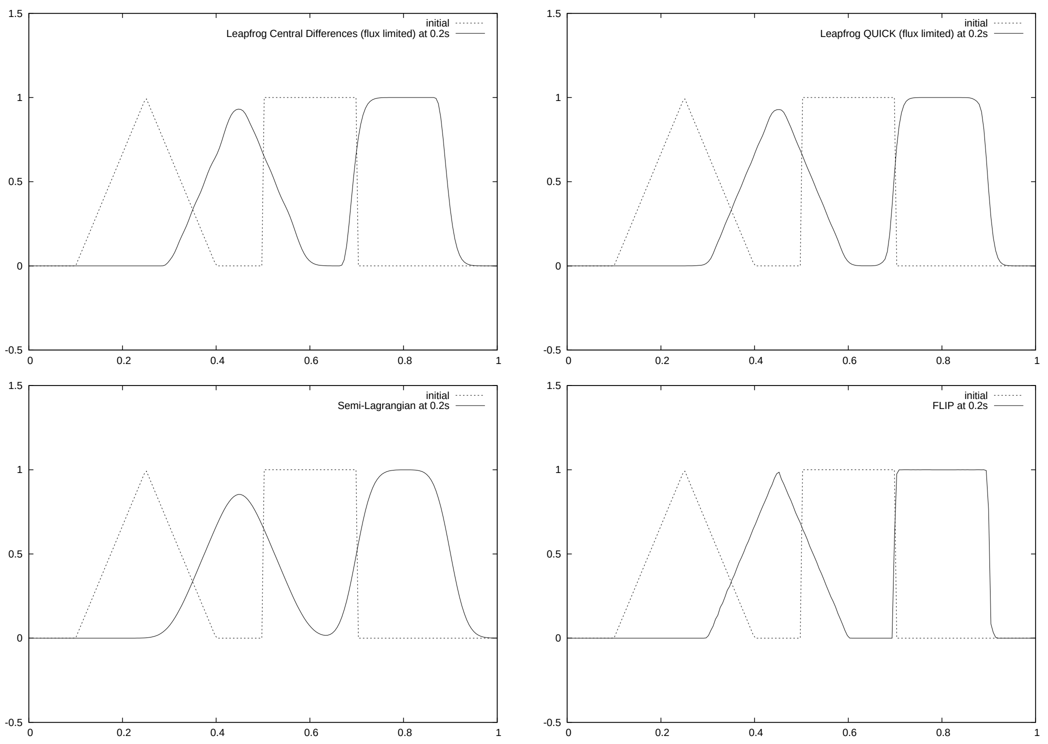

Figure III.8 Comparison of advection methods, from left to right and top to bottom: first-order differences, QUICK method, first order upwind differencing, semi-lagrangian, and the FLIP method

IV

Applications & Results

Here we present the results of our comparison simulations and a brief discussion of implementation and extensions to existing algorithms.

IV.1 Software Architecture

It is worth spending some amount of time discussing the software architecture of a fluid simulator to give an overview of the basic structure order of operations for a fluid solver. Additionally a significant increase in versatility and understandability of fluid simulation code can be achieved by employing a few domain-specific conventions and idioms.

IV.1.A Grid Storage Issues

An extremely common operation in grid-based fluid simulation methods is querying and interpolating values on a grid, primarily for velocities, but also scalar densities and marking fluid cells as full or empty. In keeping with the stateless design of our algorithms we have found it useful to define a variety of query functions and predicates to extract velocity information from a grid:

$u \leftarrow \textsc{center}(grid, i, j)$

$u \leftarrow \textsc{leftFace}(grid, i, j)$

$u \leftarrow \textsc{rightFace}(grid, i, j)$

$u \leftarrow \textsc{topFace}(grid, i, j)$

$u \leftarrow \textsc{bottomFace}(grid, i, j)$

$u \leftarrow \textsc{interpolate}(grid, i, j, s, t)$

There is an advantage to structuring our predicates to represent *logical* information about a grid, because this insulates us from implementation changes. For example, a staggered velocity grid may implement the LEFTFACE function simply by returning its native value, but require interpolation for the CENTER function; whereas a collocated pressure grid, the native values would be reversed. Using the above predicates also makes the code far more readable than using raw array indexes.

It is also useful to precompute the fluid marker status of cells and supply the above predicates, which are used extensively in determining boundary conditions and pressure matrix values:

$(true, false) \leftarrow \textsc{empty}(grid, i, j)$

$(true, false) \leftarrow \textsc{full}(grid, i, j)$

$(true, false) \leftarrow \textsc{surface}(grid, i, j)$

$(true, false) \leftarrow \textsc{solid}(grid, i, j)$

Additionally two data structures that can be used to implement the above predicates were also to accelerate grid-based solvers were used extensively in our implementation, we describe them as functions, both of which can be computed in the same pass:

$[1...n] \leftarrow \textsc{particleIndices}(particles, grid, i, j)$

$[(i_1, j_1), (i_2, j_2)...(i_n, j_n)] \leftarrow \textsc{fullCells}(particles, grid)$

PARTICLEINDICIES returns a list of index values into our total array of the particle system. This acts as a grid-acceleration structure for our particle system, returning a list of indices into an ordered array of stored particles. This allows us not only to determine which cells contain fluid, but allows us to process particles by grid cell instead of iterating over a global particle list. fullCells returns a list of indices of grid cells which contain fluid. In many circumstances, a large amount of the fluid domain is empty, and only areas the areas that contain fluid is of interest; processing cells by iterating over these indices often allows us to avoid extraneous querying of empty cells.

IV.1.B The Simulation Loop

Physical simulation generally occurs by starting with an initial set of conditions, and then advancing the simulation frame by frame. Each advancement of the simulation defines one iteration in a loop. We split this simulation loop up into the following distinct phases for the marker-and-cell method:

```
advanceFluid ($markers, grid, oldGrid, \Delta t$)
1  advectMarkers($markers, \Delta t$)
2  updateGridFromMarkers($grid, markers$)
3  swap($grid, oldGrid$)
4  advectVelocity($grid, oldGrid, \Delta t$)
5  addExternalForces($grid, \Delta t$)
6  diffuse($grid, oldGrid, \Delta t$)
7  pressureProjection($grid$)
8  extendVelocitiesOutsideFluid($grid$)
9  enforceBoundaryConditions($grid$)
```

Note the swapping of grids, some grid operations require the values from the previous timestep. This can be accomplished by storing an 'old' data and a 'new' data grid, and swapping them (only a pointer swap is necessary) so that the old data can be read without being overwritten.

Another advantage here is the order of operations; pressure projection and enforcing boundary conditions is only performed once, and are the last steps. This ensures that the two operations that require it: advecting the markers and advecting the velocities, are both performed on a divergence free field with correct boundary velocities.

Another aspect to note is that when possible, algorithms are made *stateless*. This allows for greater flexibility in being able to 'plug in' different algorithms for advection, diffusion and pressure projection. This forms the foundation of the implementation of our grid based method comparison. As most fluid simulation research has occurred over a fairly large stretch of time, implementers have used a wide variety of marking schemes, or pressure projection methods, and generally any of the above steps may vary from implementation to implementation.

This framework allows us to compare one subset of algorithms presented in a variety of papers in isolation with respect to other aspects of the fluid solver. In this discussion we have chosen to focus on the advection portion of the fluid solver, because it often has the most prominent visual impact on the solver and such a large body of research has been devoted to its solution. By simply virtualizing the ADVECTVELOCITY method, we can plug in a variety of different advection methods and compare their effects, leaving the viscosity and pressure projection methods constant.

IV.2 Stabilizing Unstable Solvers

One significant problem with finite-difference based grid methods is that they are not unconditionally stable. That is, in some circumstances the simulation may diverge. It is important to note that both stable and unstable solvers contain error, but unstable solvers manifest their errors as increases in magnitude from previous timesteps, whereas stable solvers manifest their errors as damping or smoothing. Standard finite-differencing methods are often subject to what is known as the Courant-Friedrichs-Lewy, or CFL condition. Stated simply: a finite differencing method is unstable if the particle being simulated moves fast enough to cross an entire grid cell in one timestep. Or, for a method to be stable it must satisfy:

$$\frac{u \cdot dt}{\Delta x} < 1$$

If an unstable solver violates the CFL condition, the solution quickly diverges.

This can be true even with the flux limiting technique discussed in the preceding section. Flux limiting adjusts to a dissipative or stable scheme when there is sharp spatial change in velocities spatially, but not temporally. This can cause flux limited solvers to diverge when timesteps are large enough to produce a large number of CFL violations in different grid cells.

To motivate our solution to this problem, let us consider the example of simulating a simple mass/spring model. Using Hooke's law, the spring force is linear with respect to the displacement of its mass $\mathbf{f} = -k\mathbf{\Delta x}$. To simulate this mass-spring system, we compute the acceleration on the object due to this force, and integrate it successively to obtain velocity and subsequently position. Like many physical simulations, the resultant position feeds back into the computation of the force at the next timestep. In the case of spring system, consider a circumstance in which the spring displaces just a bit farther than expected, this will result in an an increased force, which will cause the spring to rebound farther in the other direction, magnifying the original error. Eventually the system will 'explode' when the result becomes totally dominated by these magnified errors.

One possible solution to the problem of diverging behavior is to simply add a damping term, in fact very few real life systems are as ideal as our mass-spring example above; consider a slightly more sophisticated spring simulation: $\mathbf{f} = -k\mathbf{\Delta x} - K\mathbf{u}$, where K is a damping constant and $\mathbf{u}$ is the velocity, thus the force is damped as the velocity increases. Our fluid simulation has an analogous concept: viscosity, and indeed when the viscosity is very large, divergent behavior is uncommon, but not completely eliminated, and we do not want to restrict our solvers to a range of viscosities.

We now present a simple method to stabilize an unstable advection solver by rectifying unstable errors with stable errors by running a primary unstable solver and a secondary stable solver in parallel. Our approach is to prevent divergent behavior by reverting to an unconditionally stable solver when the *change* in velocity as computed by the unstable solver from the previous timestep approaches some fraction of the CFL condition, analogous to when a spring begins to oscillate wildly.

This calculation is performed locally on each grid cell. This can be performed as a post-process on the advection step as follows:

```
if   $\left( \frac{|\textsc{unstableAdvect}(i,j) - u^n_{i-\frac{1}{2},j}| \cdot dt}{\Delta x} <= \alpha \right)$
    then  $u^{n+1}_{i-\frac{1}{2},j} = \textsc{unstableAdvect}(i,j)$
    else  $u^{n+1}_{i-\frac{1}{2},j} = \textsc{stableAdvect}(i,j)$
```

$\textsc{stableAdvect}(i,j)$ and $\textsc{unstableAdvect}(i,j)$ above denote the velocity after the advection step: $u_{i-\frac{1}{2},j}$ as computed by the sing the stable and unstable advection solvers respectively. By construction, when the timestep is sufficiently small relative to the CFL condition, the stabilized method produces identical results to the primary unstable solver. In practice, we have found that an $\alpha = 0.25$ prevents divergent behavior in almost all circumstances, including vigorous interactive manipulation of body forces. Additionally, reverting to the stable method is required on average in less than 1% of the cells to be computed, even for large timesteps. The stable solver also need not be run in parallel in its entirety, only the advection substep when our stricter stability condition is violated, so the computational overhead is negligible. In all of the following comparison simulations, all of the unstable schemes were stabilized using the semi-lagrangian advection scheme described in Chapter III.

IV.3 Simulating Liquids & Gases

The following figures are still frames of 2D animations depicting the fluid simulation methods discussed in the previous sections. The full fluid animations from which these frames were generated are accompanied in the distribution of this document, and provide a much more detailed picture of simulation characteristics over time. These figures should serve to give only a rough idea of the differences in the behavior of each simulation.

Figures IV.1 - IV.6 show two-dimensional solver results for liquids. The liquid is visualized by particles marking the fluid location. In these simulations, the following settings were used: grid size: 64×64, fluid domain: $1m^2$, gravity: $-9.8m/s$, timestep dt: 0.001s kinematic viscosity ν: $1.0 \times 10^{-3} Pa \cdot s$.

Figures IV.7 - IV.12 show two-dimensional solver results for simulating gas. The light colored density serves as a temperature field and to mark density of visible particles, in other words, smoke. The simulation settings are identical to those above, however the viscosity has been altered to: $1.0 \times 10^{-6} Pa \cdot s$.

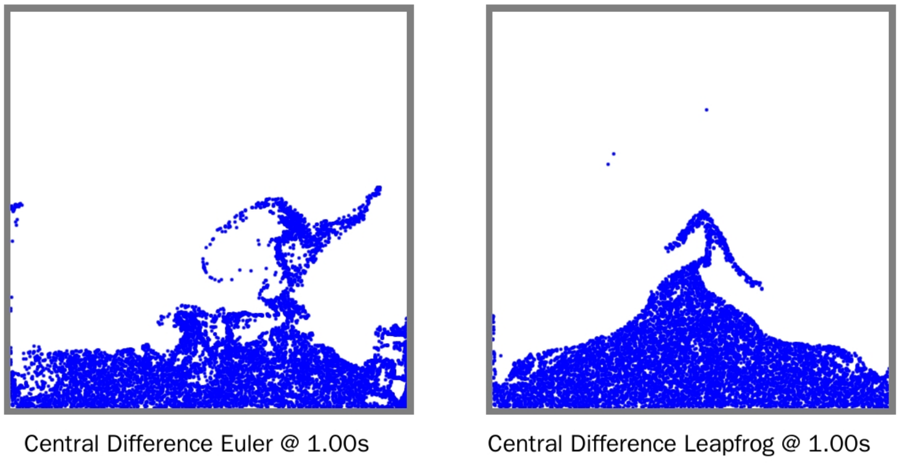
Central Difference Euler @ 1.00s
Central Difference Leapfrog @ 1.00s

Figure IV.1 Comparison of Central Differencing advection method without using forward euler integration (left) and leapfrog integration (right).

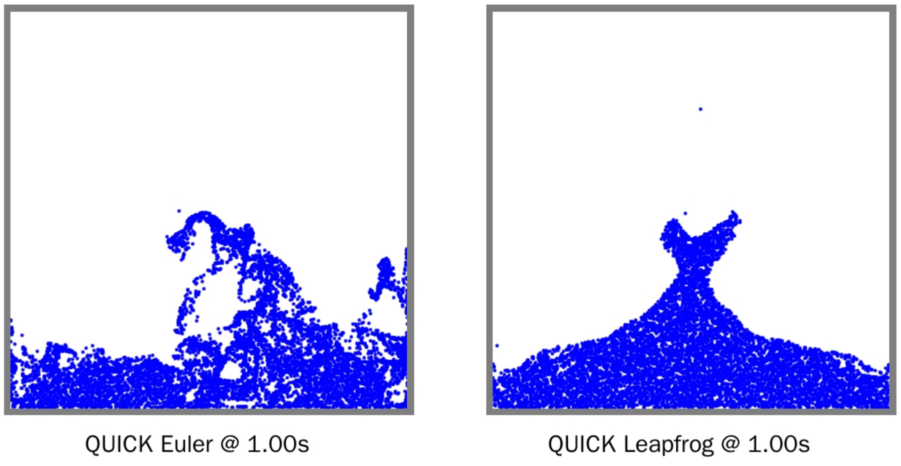

Figure IV.2 Comparison of QUICK advection method without using forward euler integration (left) and leapfrog integration (right).

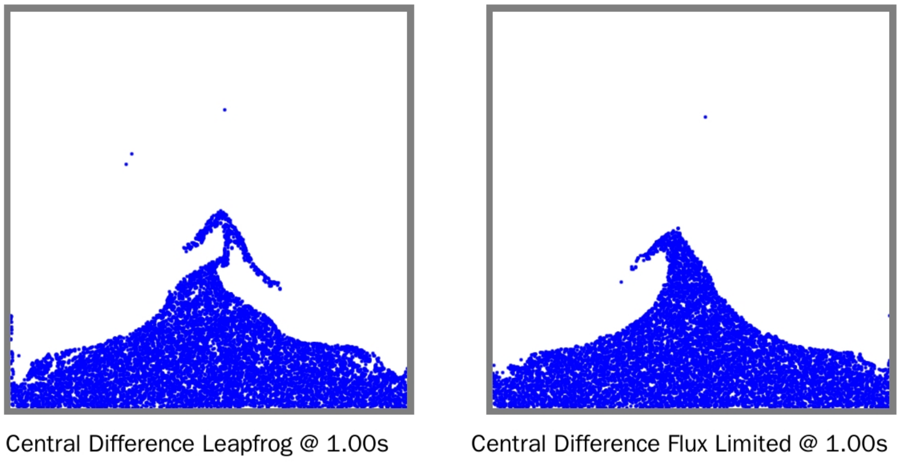
Central Difference Leapfrog @ 1.00s
Central Difference Flux Limited @ 1.00s

Figure IV.3 Comparison of Central Differencing advection method without using flux limiting (left) and with flux limiting (right) .

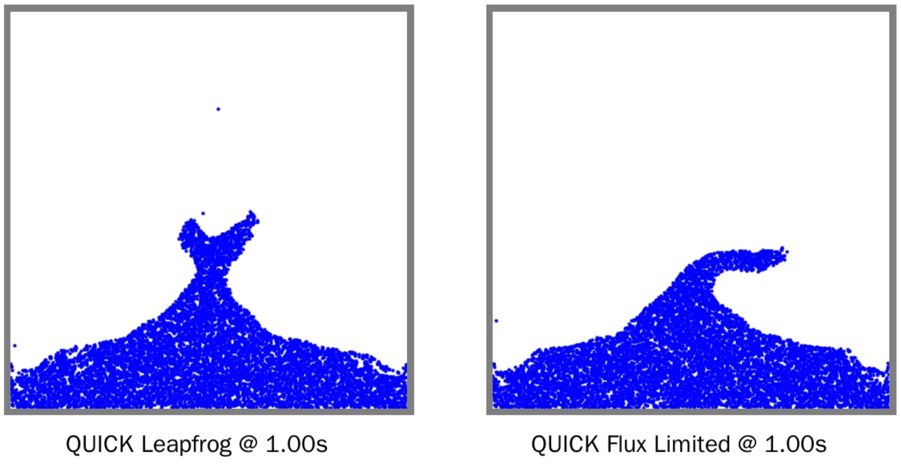

Figure IV.4 Comparison of QUICK advection method without using flux limiting (left) and with flux limiting (right) .

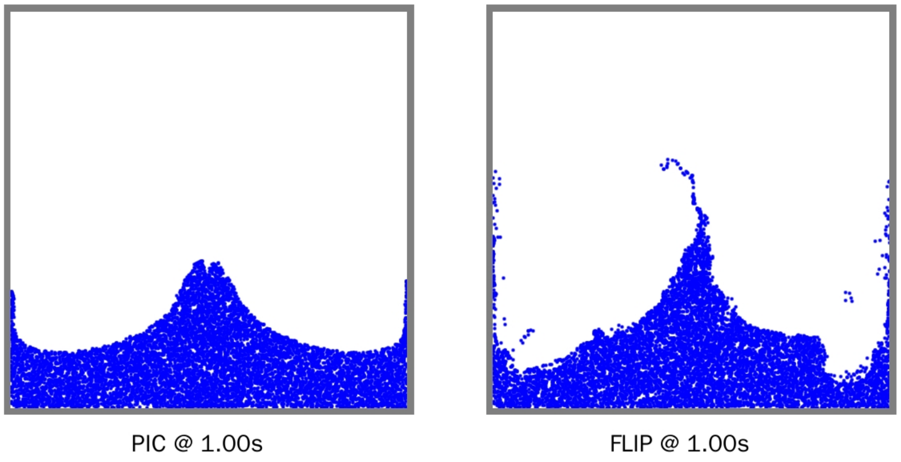

Figure IV.5 Comparison of PIC advection method (left) and the FLIP method (right) .

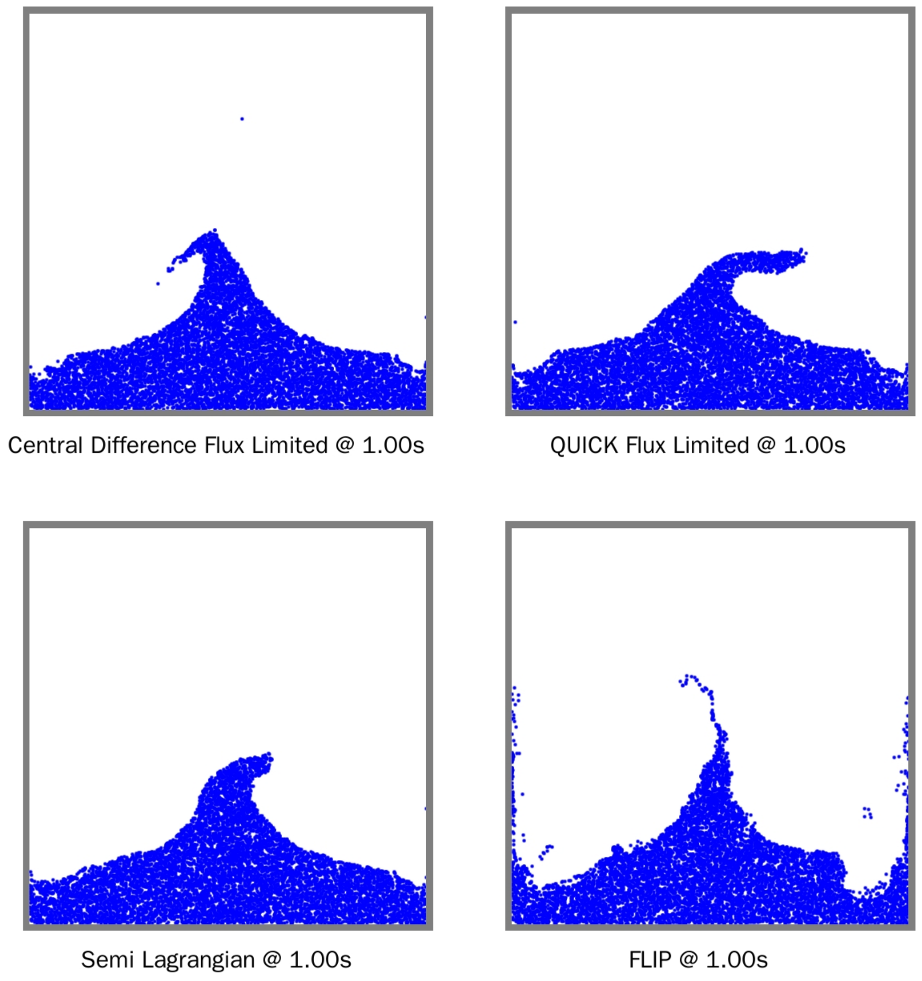
Central Difference Flux Limited @ 1.00s
QUICK Flux Limited @ 1.00s
Semi Lagrangian @ 1.00s
FLIP @ 1.00s

Figure IV.6 Comparison of the four advection methods: Central Differences, QUICK, Semi Lagrangian, and FLIP

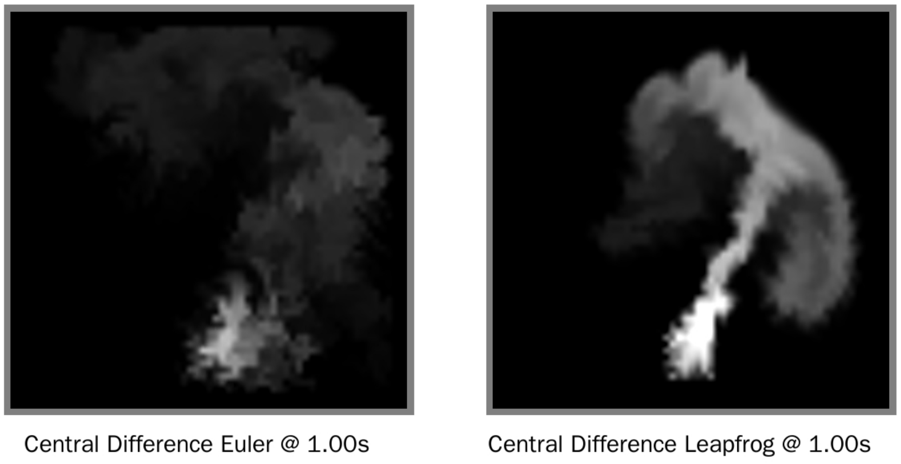
Central Difference Euler @ 1.00s
Central Difference Leapfrog @ 1.00s

Figure IV.7 Comparison of Central Differencing advection method without using forward euler integration (left) and leapfrog integration (right).

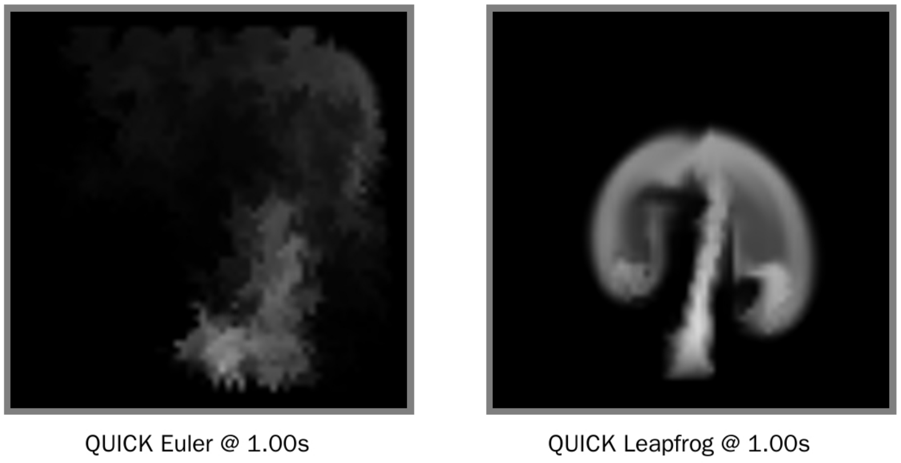

Figure IV.8 Comparison of QUICK advection method without using forward euler integration (left) and leapfrog integration (right).

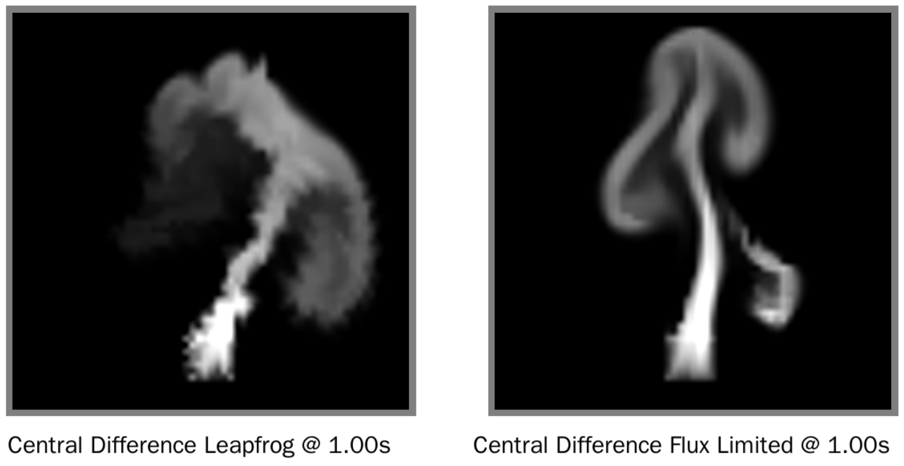
Central Difference Leapfrog @ 1.00s
Central Difference Flux Limited @ 1.00s

Figure IV.9 Comparison of Central Differencing advection method without using flux limiting (left) and with flux limiting (right) .

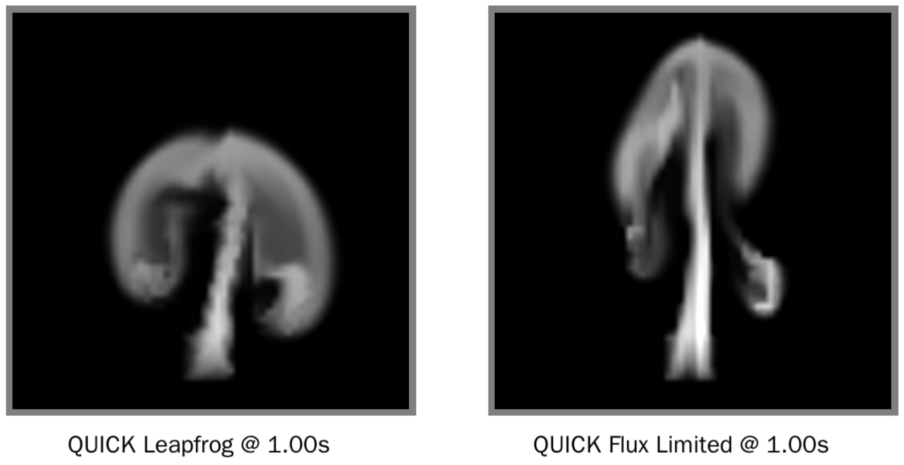

Figure IV.10 Comparison of QUICK advection method without using flux limiting (left) and with flux limiting (right) .

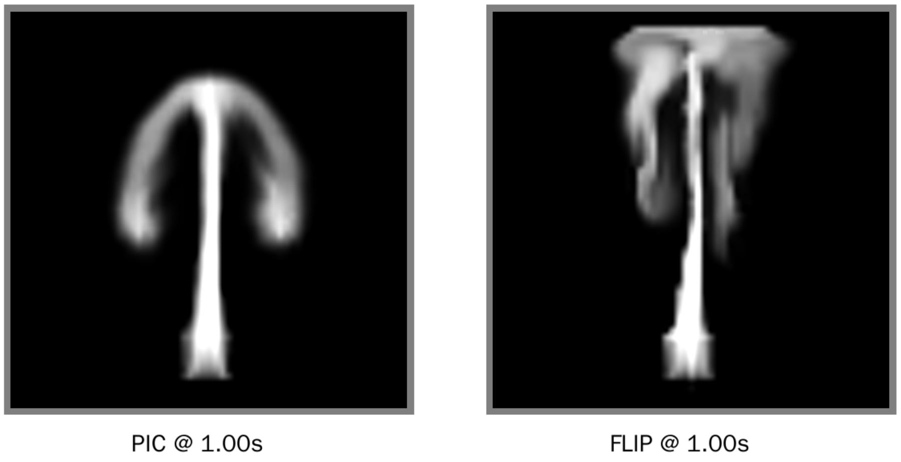

Figure IV.11 Comparison of PIC advection method (left) and the FLIP method (right) .

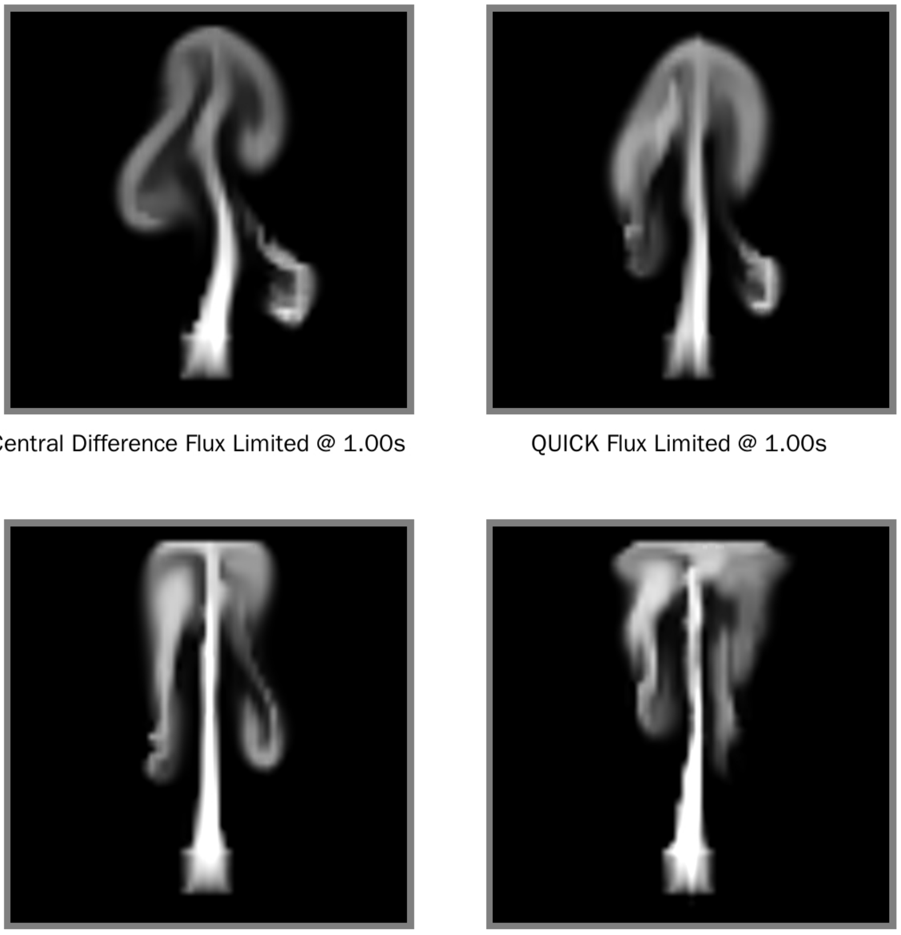
Central Difference Flux Limited @ 1.00s
QUICK Flux Limited @ 1.00s
Semi Lagrangian @ 1.00s
FLIP @ 1.00s

Figure IV.12 Comparison of the four advection methods: Central Differences, QUICK, Semi Lagrangian, and FLIP

V

Discussion of Results

V.1 Conclusion

Table V.1 shows a comparison of the stability and quality characteristics of all of the fluid solvers presented in the previous section. We conclude this section with a summary of our results and contributions.

Table V.1 A comparison of the accuracy and stability of fluid simulation algorithms Stability is analyzed based on its behavior when velocities are below and above the CFL condition: unconditionally unstable algorithms are unstable under any velocity v conditionally stable algorithms are stable when $v <= CFL$ and unstable otherwise. Unconditionally stable algorithms are stable for all v. Accuracy is defined on a spectrum between Noisy, Accurate, and Damped. .

Advection Method	**Stability**	**Accuracy**
Central Difference Euler	Uncond. Unstable	Noisy
Central Difference Leapfrog	Cond. Stable	Noisy
Central Difference Flux-Limited	Cond. Stable	Accurate
Quick Euler	Uncond. Unstable	Noisy
Quick Leapfrog	Cond. Stable	Noisy/Accurate
Quick Flux-Limited	Cond. Stable	Accurate
PIC	Uncond. Stable	Damped
FLIP	Uncond. Stable	Accurate
Semi-Lagrangian	Uncond. Stable	Damped/Accurate

One of the first important observations that should be noted is that in both the 1D plots and the 2D animations, when using forward Euler integration, *finite difference algorithms diverge*. In order to remedy this divergence, either a substantial amount of viscosity must be introduced (analogous to a spring damping coefficient), or a more stable integration method is required. In Figure IV.1, we can see a more meaningful representation of the of the wildly varying one-dimensional advection plot in Figure III.4. The simulation becomes almost completely random and divergent after one second of simulation. The leapfrog advection method, while still noisy, still retains the overall shape of the fluid. All of the simulations which use leapfrog advection are stable over time, and while they are still noisy, they don't diverge.

The second important observation from our simulation results is the value of using *flux-limiting* for finite difference solvers. In both the plots and simulated results, we can clearly see that adding the flux-limiting scheme corrects this noise and produces smooth, but not overly damped results. This makes flux-limited finite difference schemes a viable alternative to the semi-lagrangian scheme with considerably less dissipation and without a significant increase in computational cost in addition to the underlying finite differencing.

The third important result that we have obtained is that, as show in Figure IV.11, accurate and visually pleasing results can be obtained using the PIC and FLIP methods for fluid simulations without a free-surface by instantiating particles in an open fluid domain, with comparable preservation of small-scale detail as is obtained in liquid simulations. Additionally, the FLIP solver in Figures IV.6, IV.12 produce more accurate results on a coarse grid than the other grid-based solvers, due to the fact that our particles are sampling the fluid density much more highly for the purposes of solving the advection equation. The disadvantage of this is that this high particle density does incur a substantial performance penalty. An analysis of performance, memory usage, and accuracy for grid-based vs. particle based methods remains an interesting area for future work.

With these comparisons there also still does not seem to be one solver that fits best in all circumstances. The semi-lagrangian solver provides a very good tradeoff of simplicity of implementation and understanding and accurate and reliable results. The first-order finite difference scheme is very simple to implement as well and, while noisy, it may be acceptable in situations where noise is acceptable or desired (such as moderate viscosity flows, or in place of artificially added noise). Additionally, the 4th-order QUICK scheme has a wide difference stencil, examining several cells beyond those immediately adjacent. This makes it attractive for steady state flows with large grid resolutions, but on coarse grids, this can cause artifacts at fluid interfaces in which areas outside of the fluid are in the stencil.

V.2 Deficiencies of Grid Based Methods & Future Work

While grid based solvers have *volume conservation* properties that are more desirable with comparison to particle based fluid solvers, one disadvantage to grid-based solvers is that volume conservation purely indirect. Volume conservation is enforced only by constructed velocities and the behavior of the fluid marker under those velocities. If the fluid marker undersamples the grid, cells can become empty and volume can quickly be lost.

A related issue to volume conservation is that grid-based solvers have no concept of increase in pressure due to compression or addition of fluid volume into an enclosed space. While it may not be required to simulate full compressible flow, these kind of pressure increases are common cases in the real world that a general purpose fluid simulator should handle. Explicit conservation of volume and mass as an augmentation or extension to existing grid-based solvers remains an unexplored future work.

Compared to some non-grid-based methods, such as particle based fluid simulations, grid-based methods can impose a significant increase in algorithmic complexity in both time and space. This can be mitigated by making use of adaptive grids and techniques to compress cells that are uniformly solid, fluid, or empty, but this cost can still make grid-based approaches unsuitable for certain applications. A common item discussed in engineering-oriented CFD literature is running fluid simulations concurrently on multiple processors and threads of execution. This discussion is generally sidestepped by fluid simulation literature in computer graphics, but as the availability of multi- and many-core processors increases, efficient and easy-to-implement parallel algorithms for fluid simulation will be required.

One of the most active current areas of research in simulating fluids, particularly liquids, is the *level set*. Level sets as a means of marking the fluid have become increasingly popular in both academic and industry production circles, both for their ability to be directly renderable (or easily polygonizable), and their ability to accurately resolve surface and curvature. A large amount of research has already been conducted on level sets, primarily by Fedkiw et. al. and Enright et. al. as discussed in Chapter I; this has laid the groundwork for a large body of research extending level set approaches, and integrating level sets effectively with existing surface representations such polygons and points.

References

[1] John Anderson. *Computational Fluid Dynamics.* McGraw-Hill, New York, 1995.

[2] Akio Arakawa. Computational design for long-term numerical integration of the equations of fluid motion - two-dimensional incompressible flow. *J. Comput. Phys.*, 1(1):103–114, 1966.

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

[4] Mark Carlson. Phd. thesis: Rigid, melting, and flowing fluid, 2004.

[5] Mark Carlson, Peter J. Mucha, III R. Brooks Van Horn, and Greg Turk. Melting and flowing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 167–174, New York, NY, USA, 2002. ACM Press.

[6] Mark Carlson, Petere Mucha, and Greg Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. In *SIGGRAPH 2004: Computer Graphics Proceedings*, pages 377–384, New York, NY, USA, 2004. ACM Press.

[7] Jim X. Chen and Niels da Vitoria Lobo. Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graph. Models Image Process.*, 57(2):107–116, 1995.

[8] A. J. Chorin and J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics.* Springer-Verlag. Texts in Applied Mathematics 4. Second Edition., New York, 1990., 1993.

[9] James W. Daily and Donald R.F. Harleman. *Fluid Dynamics.* Addison Wesley Publishing, Reading, MA, USA, 1966.

[10] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, *Eurographics Workshop on Computer Animation and Simulation*

(EGCAS), pages 61–76. Springer-Verlag, Aug 1996. Published under the name Marie-Paule Gascuel.

[11] Dimitri Dirkakis and William Rider. *High-Resolution Methods for Incompressible and Low-Speed Flows*. Springer-Verlag, Berlin, Heidelberg, Germany, 2005.

[12] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Kuehling Pedersen. Modeling and rendering of weathered stone. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 225–234, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[13] D. Enright, D. Nguyen, F. Gibou, and R. Fedkiw. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows.

[14] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH 2002: Computer Graphics Proceedings*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[15] M. W. Evans and F. H. Harlow. The particle-in-cell method for hydrodynamic calculations. In *Los Alamos Scientific Laboratory report*, pages LA–2139., 1957.

[16] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 15–22. ACM Press, 2001.

[17] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. *ACM Trans. Graph.*, 24(3):904–909, 2005.

[18] Nick Foster and Ronald Fedkiw. Practical animations of liquids. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 23–30. ACM Press, 2001.

[19] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing: GMIP*, 58(5):471–483, 1996.

[20] Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[21] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1986. ACM Press.

[22] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical simulation in fluid dynamics: a practical introduction.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[23] Eran Guendelman, Andrew Selle, Frank Losasso, and Ronald Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph.*, 24(3):973–981, 2005.

[24] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. In *Phys. Fluids 8*, pages 2182–2189., 1965.

[25] F. H. Harlow. The particle-in-cell method for fluid dynamics. In *Methods in Computational Physics*, volume 3, New York, NY, USA, 1964. B. Alder, S. Fernbach, and M. Rotenberg, Eds .Academic Press.

[26] Mark J. Harris. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics.* Pearson Higher Education, 2004.

[27] Brackbill J.U. and H.M. Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. In *J. Comp. Phys.*, volume 65, page 314, 1986.

[28] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH 1990, Computer Graphics Proceedings*, pages 49–57, New York, NY, USA, 1990. ACM Press.

[29] A.G. Kermani M.J. Gerber and Stockie J.M. Thermodynamically based moisture prediction using roes scheme. In *The 4th Conference of the Iranian Aerospace Society.* Amir Kabir University of Technology, 2003.

[30] Lucy L.B. A numerical approach to the testing of the fission hypothesis. *Astronomy Journal*, 82:1013–1024, dec 1977.

[31] B. P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comput. Methods Appl. Mech. Eng.*, pages 59–98, 1990.

[32] M. N. Gamito E F. Lopes and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *In Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation*, pages 3–15. Springer-Verlag, 1995.

[33] F. Losasso, R. Fedkiw, , and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computational Fluids*, 2005.

[34] Frank Losasso, Frederic Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.*, 23(3):457–462, 2004.

[35] Antoine McNamara, Adrien Treuille, Zoran Popovic;, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, 2004.

[36] Matthias Mueller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[37] Ulrich Ruede Nils Thuerey. Free surface lattice-boltzmann fluid simulations with and without level sets. In *Vision, Modeling and Visualization Proceedings*, 2004.

[38] Elcott S. Tong Y. Kanso E. Schroeder P. and Desbrun M. Discrete, circulation-preserving, and stable simplicial fluids. *Preprint, Caltech*, 2005.

[39] Frederic Pighin, Jonathan M. Cohen, and Maurya Shah. Modeling and editing flows using advected radial basis functions. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 223–232, New York, NY, USA, 2004. ACM Press.

[40] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 193–202, New York, NY, USA, 2004. ACM Press.

[41] Nick Rasmussen, Duc Quang Nguyen, Willi Geiger, and Ronald Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph.*, 22(3):703–707, 2003.

[42] Andrew Selle, Alex Mohr, and Stephen Chenney. Cartoon rendering of smoke animations. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 57–60, New York, NY, USA, 2004. ACM Press.

[43] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, 2005.

[44] Jos Stam. Stable fluids. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles, 1999. Addison Wesley Longman.

[45] Jos Stam. Real-time fluid dynamics for games, 2003.

[46] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 369–376, New York, NY, USA, 1993. ACM Press.

[47] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis, 2nd Ed.* Springer-Verlag, Berlin, Germany; New York, U.S.A., 1991.

[48] Jerry Tessendorf. Simulating ocean water. In *SIGGRAPH '02: Proceedings of the conference on SIGGRAPH 2002 course notes*, page 19, New York, NY, USA, 2002. ACM Press.

[49] Adrien Treuille, Antoine McNamara, Zoran Popovic;, and Jos Stam. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22(3):716–723, 2003.

[50] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005.

[51] David J. Webb, Beverly A. de Cuevas, and Catherine S. Richmond. Improved advection schemes for ocean models. *Journal of Atmospheric and Oceanic Technology*, pages 1171–1187, 1998.

[52] Yongning Zhu. Masters thesis: Animated sand as a fluid, 2005.

[53] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.