RIGID, MELTING, AND FLOWING FLUID

A Thesis
Presented to
The Academic Faculty

by

Mark Thomas Carlson

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
August 2004

RIGID, MELTING, AND FLOWING FLUID

Dr. Greg Turk, Adviser

Dr. Peter J. Mucha, Co-Adviser
(School of Mathematics)

Dr. Irfan Essa

Dr. James F. O'Brien
(University of California at Berkeley)

Dr. Andrzej Szymczak

Date Approved: Paperwork Pending

for my fiancé, Kristi Lee Burns, who thought that staying in school and becoming fake doctors together would be a cool idea

ACKNOWLEDGEMENTS

Much of the material in this dissertation is drawn from the two publications, “Melting and Flowing” [8] and “Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid” [7]. I am extremely grateful to my coauthors, Greg Turk, Peter Mucha and Brooks Van Horn, without whose help those articles would not exist.

Jessica Hodgins and the students in the Animation Lab–Alan Chen, Christina de Juan, Ron Matoyer, James O’Brien, Gary Yngve, and Victor Zordan–had a profound influence on my early growth as a researcher. Thank you.

I am also grateful for the many fruitful conversations with the members of the Geometry group, especially Brooks Van Horn and Eugene Zhang who I used repeatedly as sounding boards for ideas and troubles.

I would also like to thank Gabriel Brostow and the other CPL folks for their willingness to always lend a hand, even when they themselves needed extra hands to get their own work done.

Most of all, I would like to thank Greg Turk for his patience and understanding. Greg always allowed me to steer my own course with research, yet he was ever there to help navigate through the rough times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbf{1}$	The identity matrix with is zero except for the diagonal which contains ones, p. 43.
$\mathbf{A}$	$\mathtt{N} \times \mathtt{N}$ matrix where each row represents one of the $\mathtt{N}$ equations that form a linear system which together with the $\mathtt{N}$ known values, $\mathbf{b}$, will solve for the $\mathtt{N}$ unknowns in $\mathbf{x}$, p. viii.
$\mathbf{A}$	The backward Euler matrix formulation of the viscous diffusion, p. 44.
$\mathbf{A}_c$	The acceleration due to collisions, p. 62.
$\mathbf{b}$	The column vector of known values in the linear system $\mathbf{Ax} = \mathbf{b}$, p. 29.
$\mathbb{C}$	The computational domain, $\mathbb{R} \bigcup \mathbb{F}$, p. 57.
d	The dimension, d=2 in 2D, p. 41.
$\mathbf{D}$	The viscous difusion matrix, p. 43.
$\mathbf{D}[\,]$	The deformation operator, p. 57.
dy	The volume of a MAC grid cell that is occupied a solid, p. 64.
$\mathbb{F}$	The fluid domain, p. ix.
$\mathbf{f}$	The acceleration, or *body force* per unit mass in meters per second per second, p. 17.
f	The x-component of the body force, $\mathbf{f}$, p. 28.
$\mathbf{F}$	Force due to collision, p. 62.
g	The y-component of the body force, $\mathbf{f}$, p. 28.
$\tilde{\mathbf{G}}$	The pseudo-Gain of a time derivative created to help explain stability, p. 21.
h	The z-component of the body force, $\mathbf{f}$, p. 28.
$\mathbf{H}$	The matrix used to help solve the Heat eqauaion with the LOD method, p. 46.
$\mathbf{I}$	The moment of inertia of a rigid body in world coordinates, p. 62.
$\mathtt{I}$	The number of gird cells in the x-dimension of the computational domain, p. 19.
$\mathtt{J}$	The number of gird cells in the y-dimension of the computational domain, p. 19.
$\mathtt{K}$	The number of gird cells in the z-dimension of the computational domain, p. 19.
k	The thermal diffusion constant that controls how fast heat moves through a material, p. 46.
M	The mass of a rigid body is its density times its volume, p. 62.
$\mathtt{M}$	The number of cells in the area of interest around the $\phi = 0$, also called the active-set or narrow band, p. 86.

$\mathbf{n}$	The outward normal on a surface, p. 60.
$\mathbf{n}_\phi$	The outward normal of the level set, p. 83.
p	The pressure is a force per unit area, p. 16.
$\mathbb{R}$	The solid domain, p. ix.
$\mathbf{r}$	A vector pointing from the center of mass to a point on the rigid body, p. 58.
$\mathbf{R}$	The force inside a rigid body that keeps it deformation free, p. 63.
$\mathbf{S}$	A source term including relative density and collision terms, p. 62.
S	A signature function of ϕ used in reinitialization, like the sign fuction but with $0 = 0$, p. 85.
t	The scalar temperature field, p. 46.
$\mathbf{t}$	The traction force, p. 60.
$\mathbf{u}$	The velocity in meters per second. The discrete version is a vector field, p. 16.
u	The x-component of the velocity, $\mathbf{u}$, p. 18.
$\mathbf{u}$	A vector that contains the x-component of the velocities for each cell face in the MAC grid that has fluid on both sides, p. 43.
$\mathbf{u}^{temp}$	The intermediate velocity used with operator splitting, p. 42.
$\hat{\mathbf{u}}$	A divergence free and energy conserving velocity that is not yet constrained to rigid body motion, p. vi.
$\hat{\mathbf{u}}_R$	The energy conserving, deformation free rigid body velocity, p. 63.
$\mathbf{u}^n$	The velocity at the beginning of a time step, p. vi.
$\mathbf{u}^{n+1}$	The final velocity at the end of a time step, p. vi.
$\mathbf{u}^*$	The divergence free velocity, without accounting for rigid body energy and motion, p. vi.
$\mathbf{u}_t$	The time derivative of velocity, p. 16.
$\tilde{\mathbf{u}}$	The best guess velocity, before enforcing incompressibility, p. 27.
$\tilde{u}$	The x-component of the best guess velocity, $\tilde{\mathbf{u}}$, p. 28.
v	The y-component of the velocity, $\mathbf{u}$, p. 18.
$\mathbf{v}$	The translational velocity of a rigid body at the center of mass, p. 58.
$\hat{\mathbf{v}}$	The rigid body velocity obtained by integrating over the rigid body's domain, p. 64.
$\tilde{v}$	The y-component of the best guess velocity, $\tilde{\mathbf{u}}$, p. 28.
w	The z-component of the velocity, $\mathbf{u}$, p. 18.

w	A number between 0 and 1 representing the fraction of volume of a computational cell that is occupied by a solid, p. 62.
$\mathbf{w}$	The characteristic function used to decide what difference opperator to use when upwinding, p. 85.
$\tilde{w}$	The z-component of the best guess velocity, $\tilde{\mathbf{u}}$, p. 28.
$\mathbf{x}$	The column vector of unknown values in the linear system $\mathbf{Ax} = \mathbf{b}$, p. 29.
$\mathbf{x}$	The center of mass of a rigid body, p. 58.
$\mathbf{y}$	A point on a rigid body, p. 58.
$\dot{\mathbf{y}}$	The velocity at a point on a rigid body, p. 58.
α_c	The angular acceleration due to collisions, p. 62.
Δt	The time step in seconds, p. 20.
$\Delta\tau$	The fictitious time step taken when smoothing the level set, p. 86.
Δx	The width of a grid cell, p. 18.
ε	A variable used to control when the smoothing should be stopped when reinitializing ϕ, p. 86.
∇	The gradient operator, p. 16.
$\nabla\cdot$	The divergence operator, p. 17.
∇^2	The Laplacian operator, p. 17.
ν	The kinematic viscosity, p. 16.
ω	The rotational velocity of a rigid body at the center of mass about the axis $\omega/\lvert\omega\rvert$ with magnitude $\lvert\omega\rvert$, p. 58.
$\hat{\omega}$	The rigid body rotational velocity obtained by integrating over the rigid body's domain, p. 64.
$\partial\mathbb{R}$	The part of a solids boundary that is touching fluid, p. 57.
ϕ	A level set, p. 65.
ϕ_x^-	The backward difference version of the first derivative of ϕ in the x-direction, p. 85.
ϕ_x^+	The forward difference version of the first derivative of ϕ in the x-direction, p. 84.
Π	The deformation stress, p. 60.
ρ	The density mass per unit volume, which for water is $1.0g/cm^3$, p. 16.
ρ_f	The density of the fluid, p. 59.
ρ_r	The density of the solid, p. 59.

Σ The extra stress tensor that depends on the deformation rate and deformation history of the fluid at a specific location, p. 59.

τ The viscous stress tensor, p. 54.

SUMMARY

This work focuses on the simulation of fluids as they transition between a solid and a liquid state, and as they interact with rigid bodies in a realistic fashion. There is an underlying theme to my work that I did not recognize until I examined my body of research as a whole. The equations of motion that are generally considered appropriate only for liquids or gas can also be used to model solids. Without adding extra constraints, one can model a solid simply as a fluid with a high viscosity. Admittedly, this representation will only get you so far, but this simple representation can create some very nice animations of objects that start as solids, and then melt into liquid over time. Another way to represent solids with the fluid equations is to add extra constraints to the equations. I use this representation in the parts of this work that focus on the two-way coupling of liquids with rigid bodies. The coupling affects both how the liquid moves the rigid bodies, and how the rigid bodies in turn affect the motion of the fluid. There are three components that are needed to allow solids and fluids to interact: a rigid body solver, a fluid solver, and a mechanism for the coupling of the two solvers.

The fluid solver used in this work was presented in [8]. This *Melting and Flowing* solver is a fast and stable system for animating materials that melt, flow, and solidify. Examples of real-world materials that exhibit these phenomena include melting candles, lava flow, the hardening of cement, icicle formation, and limestone deposition. Key to this fluid solver is the idea that we can plausibly simulate such phenomena by simply varying the viscosity inside a standard fluid solver, treating solid and nearly-solid materials as very high viscosity fluids. The computational method modifies the Marker-And-Cell algorithm [99] in order to rapidly simulate fluids with variable and arbitrarily high viscosity. The modifications allow the viscosity of the material to change in space and time according to variation in temperature, water content, or any other spatial variable. This in turn allows different locations in the same continuous material to exhibit states ranging from the absolute rigidity or slight bending of hardened wax to the splashing and sloshing of water.

The coupling that ties together the rigid body and fluid solvers was presented in [7], and is known as the *Rigid Fluid* method. It is a technique for animating the interplay between rigid bodies and viscous incompressible fluid with free surfaces. Distributed Lagrange multipliers are used to ensure two-way coupling that generates realistic motion for both the solid objects and the fluid as they interact with one another. The *rigid fluid* method is so named because the simulator treats the rigid objects as if they were made of fluid. The rigidity of such an object is maintained by identifying the region of the velocity field that is inside the object and constraining those velocities to be rigid body motion. The rigid fluid method is straightforward to implement, incurs very little computational overhead, and can be added as a bridge between current fluid simulators and rigid body solvers. Many solid objects of different densities (*e.g.,* wood or lead) can be combined in the same animation.

The rigid body solver used in this work is the impulse based solver, with shock propagation introduced by Guendelman et al. in [36]. The rigid body solver allows for collisions ranging from completely elastic, where an object can bounce around forever without loss of energy, to completely inelastic where all energy is spent in the collision. Static and dynamic frictional forces are also incorporated. The details of this rigid body solver will not be discussed, but the small changes needed to couple this solver to interact with fluid will be.

When simulating fluids, the fluid-air interface (*free surface*) is an important part of the simulation. In [8], the free surface is modelled by a set of marker particles, and after running a simulation we create detailed polygonal models of the fluid by splatting particles into a volumetric grid and then render these models using ray tracing with sub-surface scattering. In [7], I model the free surface with a particle level set technique [14]. The surface is then rendered by first extracting a triangulated surface from the level set and then ray tracing that surface with the Persistence of Vision Raytracer (http://povray.org).

CHAPTER I

INTRODUCTION

"NATURE IS THE ULTIMATE SOURCE OF INSPIRATION FOR COMPUTER GRAPHICS AND ANIMATION."

Craig W. Reynolds [71]

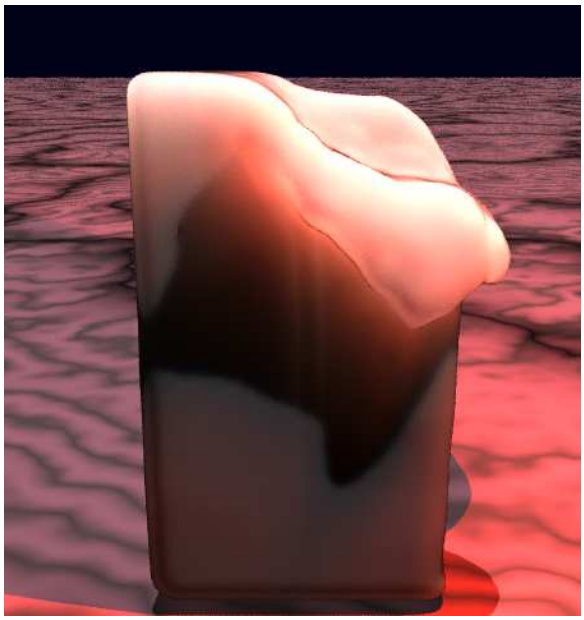

Figure 1: Melting wax.

A major goal in animation research is to simulate the behavior of real-world materials, including such phenomena as draping cloth, breaking glass, nuclear explosions, colliding rigid bodies, and flowing and splashing liquids. This thesis has two main animation research contributions: the simulation of melting and flowing materials presented in chapter 4, and the simulation of coupled rigid body and fluid motion presented in chapter 5. Both contributions will be introduced here, then chapter 2 will cover previous work in fluids and in coupling of fluids with solids. Chapter 3 will detail the implementation of a fluid simulator for computer graphics, and it covers the background material necessary to understand both the Melting and Flowing research and the Rigid Fluid research

detailed in the next two chapters. Chapter 6 will describe some important lessons learned when dealing with the free surface where the fluid meets the air.

1.1 Melting and Flowing

The goals of the work presented in chapter 4 are twofold: first, to develop a method for quickly simulating highly viscous liquids with free surfaces; second, to use this capability to animate materials that melt, flow, and harden.

The viscosity of a fluid describes how quickly the variations in the velocity of the fluid are damped out. Viscous fluids are everywhere: toothpaste, hand lotion, yogurt, ketchup, tar, wet cement, and glue are just a few examples. We have modified the Marker-And-Cell (MAC) method from the computational fluid dynamics literature to incorporate an implicit scheme for calculating the diffusion component of the equations for viscous fluids. This implicit approach allows us to take large time-steps even when the viscosity of the fluid is extreme.

The method that we present allows us to simulate material that varies in both space and time from absolutely rigid (treated as very high viscosity) to freely flowing (low viscosity). Other published methods for liquids in the graphics literature require small time-steps even for moderate viscosity, and simply cannot animate highly viscous liquids [24, 25], or they treat the fluid as inviscid and ignore the diffusive effects of viscosity all together [16, 17].

Many materials exhibit variable viscosity depending on properties such as temperature and water content. With the ability to simulate highly viscous fluid, the second goal of chapter 4 is within reach, which is to simulate materials that melt, flow, and harden. Many natural materials exhibit these properties, including wax (Figure 1), glass, cement, wet sand, stone (lava), and water (ice). To achieve this goal, the viscosity of the animated material must vary from one position to the next, and the material equations of motion are changed to address this variability. In the simulation, the viscosity becomes a function of material properties such as temperature or water content, and these varying properties allow melting and hardening. Heat advection and diffusion are simulated for the animation of material such as molten wax. The overarching theme of the melting and flowing work is that many materials that melt, flow and harden can be viewed *always* as a fluid, even when in solid form.

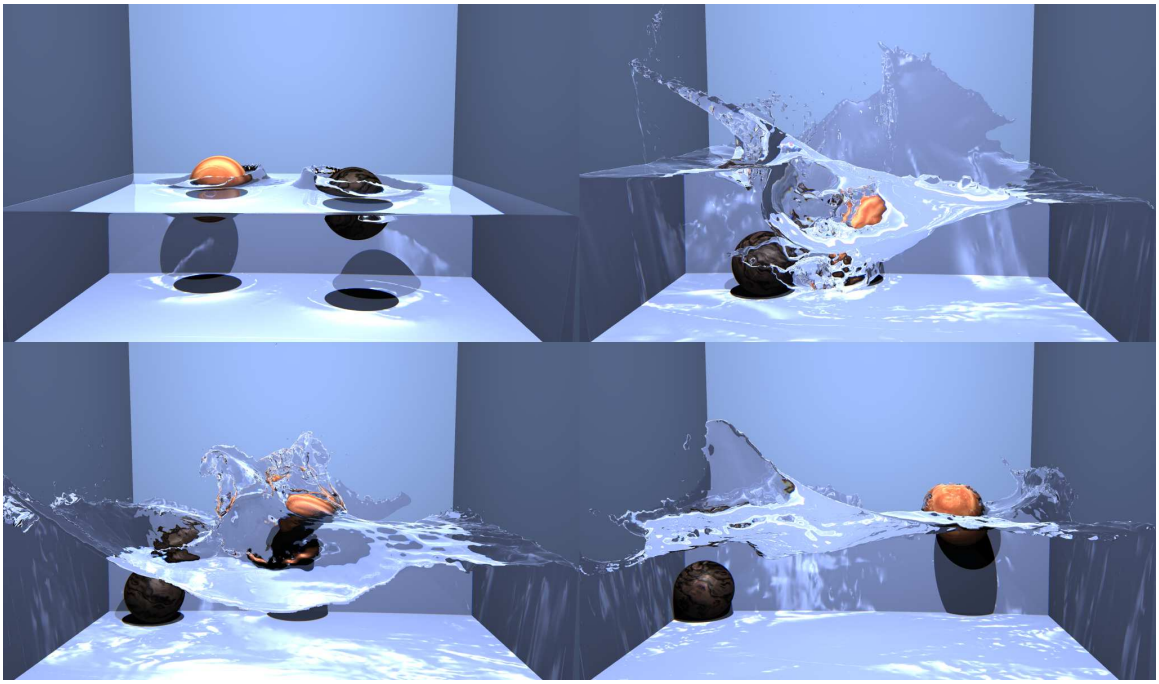

Figure 2: A lead and wood ball are thrown into a tank of water.

The melting and flowing research presented in [8] does not use a level set to define the surface, instead Greg Turk extracted the surface models from the the simulation by splatting the marker particles into a high resolution volume and then he extracted polygons from the volume. Brooks Van Horn used ray tracing with subsurface scattering to render wax and paste-like materials. The splatting and rendering will not be covered in this thesis, and but I encourage the interested reader to refer to [8] for details.

1.2 Rigid Fluid

The rigid fluid method described in chapter 5 continues with and expands on the idea that solids can be treated as fluid. Solids in the previous section were simulated as fluids with extremely high viscosity, but the rigid fluid technique treats solids as fluid with no viscous effects at all, and makes the solids rigid by enforcing extra constraints. The focus of the rigid fluid technique is to capture the complex and subtle interactions of rigid bodies and fluids.

1.2.1 Types of Coupling

Solid objects interact with fluids every day–our children play with rubber duckies in the tub, athletes dive into swimming pools, and ice clinks in our glass as we pour in our delicious Tang. This research will distinguish between three types of interaction, or coupling, that the solids and fluid can have: one-way solid-to-fluid coupling, one-way fluid-to-solid coupling, and two-way coupling. More information on coupling in a more general animation framework can be found in [56].

It is common in computer animation to see a ball splash into a pool of liquid [26, 24, 16]. This is an example of one-way solid-to-fluid coupling where the motion of the ball is predetermined and the fluid motion is a secondary effect in response to the ball. In such simulations, the fluid has no effect on the motion path of the ball, but the ball can splash the water all around.

In the other type of single direction coupling, one-way fluid-to-solid coupling, the fluid moves the solid without the solid affecting the fluid. Foster and Metaxas demonstrate this type of coupling by animating tin cans floating on top of swelling water [25]. In this type of one-way coupling the tin can could shrink to the size of a cork or grow to the size of a barrel without affecting the motion of the water.

The last type of coupling is two-way coupling. With two-way coupling of solids and fluid, simulation alone can drive many scenes that once required assistance from hand animation. For example, flood waters could sweep away a score of horseback riders, washing around them before they can reach the safety of a hastily built wall of stones, the flood water slowing only briefly as it breaks through and washes away the makeshift barrier. Alternatively, a doomed battleship, cracked in half by torpedoes, would list and sink realistically, causing eddies and whirlpools, possibly taking a few unfortunate seamen down with the undertow.

The work described in chapter 5 focuses on two-way coupling of rigid bodies and incompressible fluid. Two-way coupling of this type is in general a difficult problem [21], but with the *rigid fluid* method, two-way coupling between fluid and rigid bodies is a straightforward addition to a fluid and rigid body solver, and the extra computational cost scales linearly with the number of rigid bodies. More precisely, the computational cost of the two-way coupling, ignoring the cost of the fluid and rigid body solvers, is $O(N)$ where N is the number of Eulerian grid cells occupied by rigid

bodies.

By changing the density of a ball, the rigid fluid method can achieve vastly differing effects in the ball-splashing-into-liquid animation. If the ball is made of lead it will create a large splash and rapidly sink to the bottom, but if the ball is made of wood it will create a smaller splash, float to the surface of the liquid and bob about a bit (see Figures 2 and 19).

CHAPTER II

INFLUENCES FROM THE PAST

"IF I HAVE SEEN FARTHER THAN OTHERS, IT IS BECAUSE
I WAS STANDING ON THE SHOULDERS OF GIANTS."

Isaac Newton

Figure 3: Terminator®3: Rise of the Machines(tm), ©2003 IMF Internationale Medien un Film GmbH & Co. 3 Productions KG All Rights Reserved. Image courtesy of Industrial Light & Magic.

2.1 Previous Work in Fluids

Animation of fluids is approached in a number of ways in the computer graphics literature. We use the term *fluids* to encompass the motion of gases such as air (including simulating smoke), and liquids such as water.

Several graphics researchers studied the large-scale motion of water in waves [28, 62]. These methods use elevation maps of the terrain underneath the water, and the line of waves is bent according to the variations in wave speed that the elevation profile induces. The simulation of breaking waves occurs at a particular sea floor elevation and wave velocity. Hinsinger et al. [38] uses an adaptive mesh and a procedural approach to animate and render deep water waves at interactive rates.

Kass and Miller take a different approach to the simulation of fluids [44]. Like most of the earlier approaches, they use a height field to model water. In contrast to other methods, however,

they use a partial differential equation (PDE) formulation for the motion of the water. Their PDE's govern the amount of fluid that passes between columns of water. O'Brien and Hodgins use a hybrid height field and particle-based representation to simulate splashing water [55].

Several groups of researchers have used physically-based particle models to represent fluids. Miller and Pearce create solids, deforming objects and fluids by tuning the manner in which particles interacted with one another [50]. Their particle forces are similar to Lennard-Jones forces: particles very close together repel one another, but at moderate distances they are attracted to each other, with the attraction falling off with greater distances. Tonnesen, in addition to calculating inter-particle forces, uses a discrete approximation to the general heat transfer equation in order to modify a particle's behavior according to its thermal energy [91]. A similar approach is used by Terzopoulos et al., but they also allow pairs of particles to be explicitly attached to one another for modeling deformable objects [89]. Desbrun and Gascuel also use Lennard-Jones style particle forces to create soft materials, but they maintain an explicit blending graph and perform particle size calculations in order to preserve volume [13]. Stora et al. use particles and an approach to force calculations called *smoothed particle hydrodynamics* (SPH) in order to simulate the flow of lava [84]. Their simulator models heat diffusion and variable viscosity, and they demonstrate animations that use up to 3,000 particles. Müller et al. [53] also use a SPH based approach to simulate free surface flows, and they animate up to 5,000 particles at interactive rates. Premoe et al. [67] use a *Moving Particle Semi-implicit* (MPS) technique to simulate 10,000 particles at interactive rates, and also run non-interactive simulations with up to 150,000 particles using their technique.

A common technique in computer graphics is to simulate the incompressible Navier-Stokes equations on a fixed Eulerian grid, instead of the Lagrangian based particle approach. The first use of this *computational fluid dynamics* (CFD) approach for graphics was Chen and da Vitoria Lobo [9]. Their technique uses a relaxation technique to solve the 2D Navier-Stokes equations, and they push a height field up or down based on the pressure to create the third dimension. Witting demonstrates a system in which the 2D Navier-Stokes equations are used in an animation environment [101]. His system allows animators to create and control 2D effects such as water swirling and smoke rising. Witting uses a set of governing equations that includes heat diffusion and thermal buoyancy, and he uses a fourth-order Runge-Kutta finite differencing scheme for solving the equations. Solving

the 3D Navier-Stokes equations was popularized in computer graphics by Foster and Metaxis. In a series of several papers, they demonstrate how the Marker-And-Cell (MAC) approach of Harlow and Welch [37] can be used to animate water [25], animate smoke [27], and be augmented to control the behavior of animated fluids [26]. A major strength of their method is that liquid is no longer constrained to be a height field, as demonstrated by their animations of pouring and splashing. Stam uses a semi-Lagrangian method for fluid convection and an implicit integrator for diffusion so that large time steps can be used for animating smoke with no internal boundaries [79]. Stam was also the first in computer graphics to use the now popular Chorin based pressure projection method [10] to satisfy the zero divergence condition. Fedkiw and Stam improve upon this method using vorticity confinement to maintain the high frequency swirling in smoke, and by using clamped cubic interpolation to prevent the dissipation of fine features [20]. Their improved technique allows solid boundaries, moving or stationary, but assumes a zero viscosity fluid [20]. Shi and Yu [76] adapt a similar semi-Lagrangian smoke simulator to an adaptive octree grid so that large simulations can be run in a small amount of memory. Treuille et al. add a level of user defined key-frame control to smoke by solving an optimal control problem over every frame of the animation [94]. Stam improves on his 2D fluid simulator by allowing it to simulate flows on a Catmull-Clark surface inbeded in 3D space [82], and has a large body of work in fluid simulations for graphics including smoke in video games [83], spectral technique solvers [81], real time interactions [80].

Foster and Fedkiw re-visit the Marker-And-Cell method, and improve upon it in several ways [24]. First, they replace the forward Euler convection calculations with a semi-Lagrangian approach for greater stability. Second, and perhaps more important, they introduce and use the level set approach mixed with Lagrangian particles to computer graphics for the purpose of fluid simulations. Their level set approach results in more finely resolved details on the liquid's surface. Their technique is improved upon by adding another layer of particles outside the water. This new technique, called the particle level set, is used by Enright et al. [16] to create free surface flows with thin sheet splashes and better fluid volume preservation.

Yngve et al. demonstrate the animation of explosions using CFD based on the equations for compressible, viscous flow [105]. Their method takes care to properly model the shocks along blast wave fronts, and also models the interaction between the fluids and solid objects.

Feldman et al. animate suspended particle explosions using the incompressible fluid equations [22]. They model gas expansion and combustion by allowing nonzero values on the right hand side of the continuity equation (Equation 1).

Kunimatsu et al. [45] mention using a combined *Volume of Fluid* (VOF) and *Cubic Interpolated Propagation* (CIP) technique to simulate free surfaces. The free surfaces are delineated by a subdivision surface, and triangulated representations of that surface are used to quickly render caustic textures for realistic liquid lighting. Hong and Kim [40] use a VOF technique to simulate a two-phase fluid flow with bubbles. A polygonal surfaces is extracted from the VOF indicator function and used to compute surface tension forces and in rendering for the bubbles. Takahashi et al. [87] use a combined VOF, CIP, and particle system to simulate fluid with splash–particles that are under the effect of gravity and foam–particles that stick to the fluid surface.

Wei et al. [95] use *Cellular Automata* (CA) to simulate melting of volumetric solids at interactive rates. The transition between solid and liquid is controlled by the transfer of heat between cells. Wei et al. [97] use a technique based on the CA model called a Lattice Boltzmann Model (LBM) to animate bubbles and feathers drifting in the air. A LBM approach is used again by Wei et al. [96] to simulate the microscopic behavior of fluid so that the averaged macroscopic movement obeys the Navier-Stokes equations. They also model the buoyancy forces of gas with a temperature field.

Wrenninge wrote an overview of implementing a fluid solver for use in the visual effects industry [103]. In his thesis he describes and uses the same basic implementation as [16] but improves the definition of solid boundaries by defining them as a static level set. A sketch of Wrenninge's work, co-authored with Roble from Digital Domain, can be found in [104]. Another Sketch from Houston et al. [41] from Frantic Films describes a *Unified Occlusion Field* that stores slip conditions, velocities, and a level set boundary to represent all complex solid objects in the simulation grid. The melting and flowing research described in chapter 4 has practical uses in the visual effects industry as well. Fält and Roble from Digital Domain modified the solver in [8] for use in their in-house fluid simulator to animate fluids with extreme viscosity [23]. Sumner et al. [85] from Industrial Light & Magic (ILM) built a proprietary implicit variable viscosity simulator to melt the liquid metal skin off the Terminatrix depicted in figure 3; their method stores texture information in particles to allow skin textures of Kristanna Loken to slough off realistically. Rasmussen et al. [69],

also from ILM, simulate large smoke phenomena like nuclear explosions in 3D by interpolating between several large 2D simulations with 3D Kolmogorov velocity fields.

There are several fluid publications in press at the time of this writing. Goktekin et al. [32] animate viscoelastic fluids, such as hair gel or pudding, by adding elastic terms to the basic Navier-Stokes equations. The elastic terms are controlled by von Mises's yield condition and a quasi-linear plasticity model. Fattal and Lischinski [19] control smoke animations by adding two terms to the standard flow equations: a smoke gathering term to prevent excessive diffusion, and a driving force to move smoke densities toward target smoke states. McNamara et al. [48] use the adjoint method to reduce the complexity in solving optimal control problems to control both liquid and smoke animations with *keyframes*. They demonstrate significant speedups from the work in [94], and considerably more control including the animation of running humanoid figures made of smoke and water. Losasso et al. [47] simulate fluids on an octree data structure that is refined to capture high scale fluid movement near obstacles, liquid free surfaces, and areas with heavy smoke density. They create a symmetric discretization of the Poisson pressure matrix to solve over the entire domain at once with a preconditioned conjugate gradient method. Rasmussen et al. [70] propose a particle control system to direct liquids. Beyond control, they expand on previous fluid techniques by adding semi-implicit treatment of the viscous stress tensor, a divergence free extension velocity, overlapping or moving simulation grids, and improved solid-liquid boundary conditions. Ihm et al. [42] enhance smoke simulations by using chemical kinetics to control an extra body force term in the momentum equation as well as a divergence control term that influences the expansion and compression of the gas. Shan et al. [75] use galilean invariance to move the computational domain towards interesting portions of an unbounded fluid simulation. This domain movements allows them to run simulations with smaller grids than are used in the traditional fixed gird animations. Pighin et al. [66] convert previously run Eulerian grid, buoyancy driven fluid simulations into a new *Advected Radial Basis Function* (ARBF) representation. Once the simulation is in the ARBF representation, they are able to edit it as if it as if it was a deformable object, this allowing animators artistic control over a previously run fluid simulation. Mihalef et al. [49] introduce the *Slice Method* which uses 2D cross sections of fluid simulations to build a wave library. They use the wave library to control 3D wave simulations that break at the time they choose and have the shape they desire. Greenwood and

House [34] add bubbles to a previously run liquid simulation. Their bubbles do not affect the simulation beyond visual enhancement, but this decoupling allows them to tweak the bubble animation without having to re-run a complicated simulation.

The CFD literature contains literally thousands of papers on simulating fluids, and there are a number of approaches such as spectral methods and finite elements that are virtually untried in computer graphics. The factors that guide researchers in selecting fluid simulation methods include: ease of programming, low computational overhead, controllability, the incorporation of obstacles, and (in the case of water and other liquids) the representation of free surfaces. Spectral methods do not easily represent complex boundaries or free surfaces, and finite element methods are computationally expensive and complex. These factors are probably important for the prevalence of finite differences methods used for computer graphics.

2.2 Previous Work in Solid Fluid Coupling

Many researchers have demonstrated one-way solid-to-fluid coupling. In [9, 25, 26, 79, 20], the rigid bodies are treated as boundary conditions with set velocities. Foster and Fedkiw [24] improve on the technique described in those previous papers by allowing the fluid to move freely along the tangent of the solids. Enright et al. also use this improvement [16].

One-way fluid-to-solid coupling is demonstrated by Chen et al. [9] and by Foster et al. [25], where solids are treated as massless marker particles that moved freely on the fluid's surface. Wei et al. [97] simulate fluid-to-solid coupling of slow moving deformable light weight objects like soap bubbles and feathers in a gaseous wind field with a Lattice Boltzmann Model. Because the objects are light weight, their effect on the fluid is ignored, but a *virtual force* from the fluid does advect and deform the objects.

To the best of our knowledge, Chen and da Vitoria Lobo [9] are the first to solve the Navier-Stokes equations in a computer graphics setting. They solve the two-dimensional equations at interactive rates, and add the third dimension with a height field based on the pressure. As mentioned in the last two paragraphs, they demonstrate both kinds of one-way coupling. In addition, they also propose tuning the velocity and pressure around objects to achieve two-way coupling, although they do not implement the idea.

O'Brien and Hodgins [55] demonstrate both types of one-way coupling by dropping a ball into water. First they calculate the forces from the ball colliding with the water to transfer the energy to the water and create waves and spray from solid-to-fluid coupling. Once the ball is in contact with the water, its center of mass stays at the surface and follows the laws of one way fluid-to-solid coupling.

Yngve et al. [105] demonstrate two-way coupling of breaking objects and compressible fluids in explosions, however their technique does not apply to incompressible fluids. To achieve fluid-to-solid coupling, Yngve et al. model forces due to hydrostatic pressure, and ignore the dynamic forces due to fluid momentum. Therefore, rigid body motion that is dominated by rotational forces cannot be reproduced with their technique. The rigid fluid technique of this thesis, on the other hand, accounts for dynamic forces that arise from the fluid momentum. Yngve et al. achieve solid-to-fluid coupling by calculating the fluid displaced by a voxelized version of the solid. If the displacement is too great they use sub-steps to smoothly displace the fluid. Unfortunately, the density of the fluid could change because the fluid in their explosion simulator is compressible. They are able to account for this compression and conserve energy in this case by adding internal energy to the fluid. However, internal energy and compressibility are not part of the incompressible fluid model used in the rigid fluid method so the solid-to-fluid coupling they use simply will not work for incompressible fluids.

Nixon and Lobb use two-way coupling to model compressible fluids and deformable thin shell objects [54]. Their work models water balloon type objects, with the compressible Navier-Stokes equations modeling the fluid and a mass-spring system modeling the balloon membrane.

Takahashi et al. [88] report two-way coupling of buoyant rigid bodies and incompressible fluids using a combined *Volume Of Fluid* and *Cubic Interpolated Propagation* system. It is not clear how they incorporate the density of the rigid bodies into the simulation. Using a regular grid, they identify any cell that is more than half filled with a rigid body as a solid boundary. They set zero Neumann boundary conditions for the pressure at these boundaries to approximate solid-to-fluid coupling, and use the pressure to find forces at the boundary that act on the solid for the fluid-to-solid coupling. Takahashi et al. [87] create a variation of this technique for water with splash and foam. They incorporate a rigid body solver with the fluid solver, setting the velocity of the fluid

inside a cell containing solid to that of the solid to achieve solid-to-fluid coupling. However, they use predefined motion, and not simulated coupled motion, to move the rigid bodies. Their rigid body motion is only augmented by the forces from the pressure. The rigid fluid simulations have no pre-defined rigid body motion—all movement is determined by the two-way coupling with no user interaction or pre-defined motion. Moreover, the fluid-to-solid coupling in [88] and [87] use the pressure from the fluid to add forces to the solid. Just as in [105], dynamic forces due to the fluid momentum are not accounted for since the hydrostatic pressure forces act in a direction normal to the rigid body's surface. Since many two-way coupled animations are dominated by dynamic fluid forces spinning and pushing the rigid bodies, these techniques simply can not reproduce the lively fluid-to-solid coupling found in the rigid fluid animations.

Génevaux et al. [30] demonstrate a type of two-way coupling between an incompressible fluid and deformable solids, with a communication interface between the two. They use the MAC method for the fluid simulation, and the marker particles add forces to the interface to complete the fluid-to-solid coupling. The deformable solids are represented as mass/spring systems, and a per-cell fluid force is summed up from the solid point masses occupying the fluid cell to complete the solid-to-fluid coupling. Because their technique uses a mass/spring system to represent deformable solids (not rigid bodies like in the rigid fluid method), they would have to make the springs extremely stiff in order to avoid deformation of the solid in fast moving animations. This would cause a stiff system and they would have to take very small time steps to simulate such animations. It is unclear how the density of the objects plays a role in their simulations.

A plethora of research on the coupling of solids and fluid exists in the physics and mathematics literature. Fedkiw uses the *Ghost Fluid* method to couple compressible fluids and deformable solids [21]. Deformable solids have also been successfully treated by Peskin with the *Immersed Boundary method* [64].

Two-way coupling between fluids and rigid solids is often accomplished in the computational physics community with the *Arbitrary Lagrangian-Eulerian* (ALE) method, introduced by Hirt et al. in [39]. The ALE method is a finite element technique and suffers from two main drawbacks. First, the computational grid must be re-meshed when the elements get too distorted, an often costly procedure. The second drawback, pointed out by Singh et al. [78], is that two layers of elements are

needed in the gap between solids as they approach one another.

Researchers studying particulate suspension flows have introduced a two-way coupled computation known as the *Distributed Lagrange Multiplier* (DLM) technique [31]. The DLM method does not suffer from the need to re-mesh. The research presented in chapter 5 uses a DLM method.

CHAPTER III

A FLUID SIMULATOR

"THE FACT THAT ONE OF TWO OTHERWISE IDENTICAL COMPUTERS HAS A COPY OF THE GAME QUAKE INSTALLED PROBABLY WILL NOT AFFECT WHETHER A SIMULATION WILL PRODUCE IDENTICAL RESULTS ON THE TWO MACHINES, BUT IT WILL BE DIFFICULT TO PROVE THIS."

T. C. Belding [5]

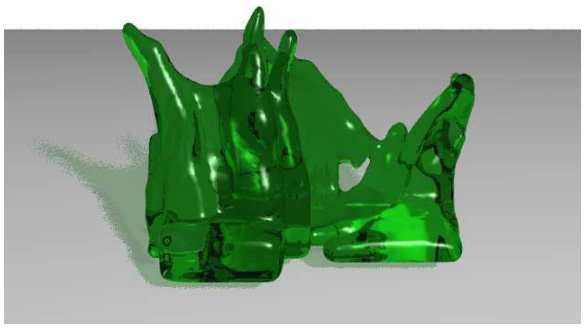

Figure 4: Splashing fluid.

The equations of motion that govern the movement of liquid are known and the *Navier-Stokes* equations. A rich history of solving the Navier-Stokes equations exists in computer animation [9, 25, 79, 98, 101, 20, 24, 8, 16], and this chapter will continue that tradition with the additional aim of supplying the reader with all the background material necessary to create their own fluid simulator. There are many choices one must make when writing a fluid simulator; the choices made when writing this solver focus on simplicity of implementation, ease of explanation, and popularity in the computer animation literature. This chapter describes one specific implementation of a fluid simulator, specifically a MAC method [25, 99] with pressure projection [79, 10] to enforce incompressibility. The chapter begins by describing the equations that govern the motion of fluid, and then it details one way to discretize those equations so that they may be solved on a discrete

computational domain. Along the way, there will be just enough background in fluid dynamics and numerical techniques that a reader should be able to reproduce the work. Be warned that the following chapter should provide the reader with enough information to be dangerous in the field of computational fluid dynamics, not necessarily knowledgable. Knowledgeable readers, who may already be dangerous, may wish to skip this chapter.

3.1 Fluid Dynamics Background

In the following discussion, the vector field $\mathbf{u}$ represents the velocity of the fluid. Pressure, a scalar field, will be represented by p, and the density of the fluid is ρ (the density is also a scalar field, but we usually take it to be 1 because the density of water is very close to 1), and the scalar field ν is the kinematic viscosity of the fluid.

The equations of motion for a viscous incompressible fluid are the Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} \; + \; \nu \nabla^2 \mathbf{u} \; - \; \frac{1}{\rho}\nabla p \; + \; \mathbf{f}. \tag{2}$$

Equation 1 states that the velocity field has zero divergence everywhere. This simply means that in any small region of fluid, the amount of fluid entering the region is exactly equal to the amount leaving the region. This is conservation of mass for incompressible fluids. In reality no fluid is ever really incompressible, but incompressibility is assumed for fluids moving at low speeds because their compression is negligible and the incompressibility allows for a convenient solution for the pressure (described below). Explosions are high speed [105], and so are jet planes moving at MACH speeds, and such phenomena are governed by other equations.

Equation 2 describes the conservation of momentum, and it has several components. Reading from left to right, it states that the instantaneous change in velocity of the fluid at a given position is the sum of four terms: advection, diffusion, pressure, and body forces. The vector field $\mathbf{u}_t$ is the time derivative of the fluid velocity. Subscript notation will be used for the partial derivative (*i.e.*, $\mathbf{u}_t = \frac{\partial \mathbf{u}}{\partial t}$). The advection term, $(\mathbf{u}\cdot\nabla)\mathbf{u}$, accounts for the direction in which the surrounding fluid pushes a small region of fluid. Fast river water is an advection-dominated flow, and any small amount of water poured into the river will quickly be swept away with the current. The momentum diffusion

term, $\nabla\cdot(\nu\nabla\mathbf{u})$, describes how quickly the fluid damps out variation in the velocity surrounding a given point. The parameter ν is the measure of kinematic viscosity of the fluid, and the higher its value, the faster the velocity variations are damped. For constant viscosity, the ν factors out yielding the more familiar momentum diffusion form, $\nu\nabla^2\mathbf{u}$. The third term, $\frac{1}{\rho}\nabla p$, is the pressure gradient, and it describes how a small parcel of fluid is pushed in a direction from high to low pressure. The final vector field term $\mathbf{f}$ contains the external forces per unit mass (called *body forces*) that act globally on the fluid; $\mathbf{f}$ is usually just gravity, but it could be the precession of the earth, the wind, or any other user-defined vector field.

To understand the Navier-Stokes equations one must also understand the differential operators used within them.

The vector differential operator ∇, pronounced "del", when used on an arbitrary scalar p creates a vector known as the gradient of p ([2], section 1.6):

$$\nabla p = (p_x, p_y, p_z). \tag{3}$$

The divergence $\nabla\cdot$ is a differential operator on a vector, for example $\mathbf{u} = (u, v, w)$, that results in a scalar ([2], section 1.7):

$$\nabla\cdot\mathbf{u} = u_x + v_y + w_z. \tag{4}$$

The Laplacian operator, ∇^2, is a scalar differential operator and is used as part of the constant viscosity diffusion equation ([11], section 4.1.1). The Laplacian operator on a scalar u is

$$\nabla\cdot(\nabla u) = \nabla^2 u = u_{xx} + u_{yy} + u_{zz}. \tag{5}$$

3.2 Computational Fluid Dynamics Background

A computer, obviously, can not solve the Navier-Stokes equations at all the infinite number of points in a continuous domain, so to solve equations 1 and 2 a discrete domain must be chosen. This section describes a discrete computational domain, commonly known as a MAC grid, and a numerical technique known as finite differences, to break the Navier-Stokes equations into bite sized pieces that the computer can solve. The MAC grid is described first. The MAC grid gets its name from the Marker-And-Cell method of simulating fluids with free surfaces which was originally described by Harlow and Welch in 1965 [37]. The MAC approach is described well in several other

publications [25, 35, 99], but in the hopes of making this a self-contained work and to fill in some deficiencies in the 3D portions of previous descriptions, I will describe one version of the method here. First I will define the MAC grid and the finite differences background necessary to understand the Marker-And-Cell technique.

3.2.1 The MAC Grid

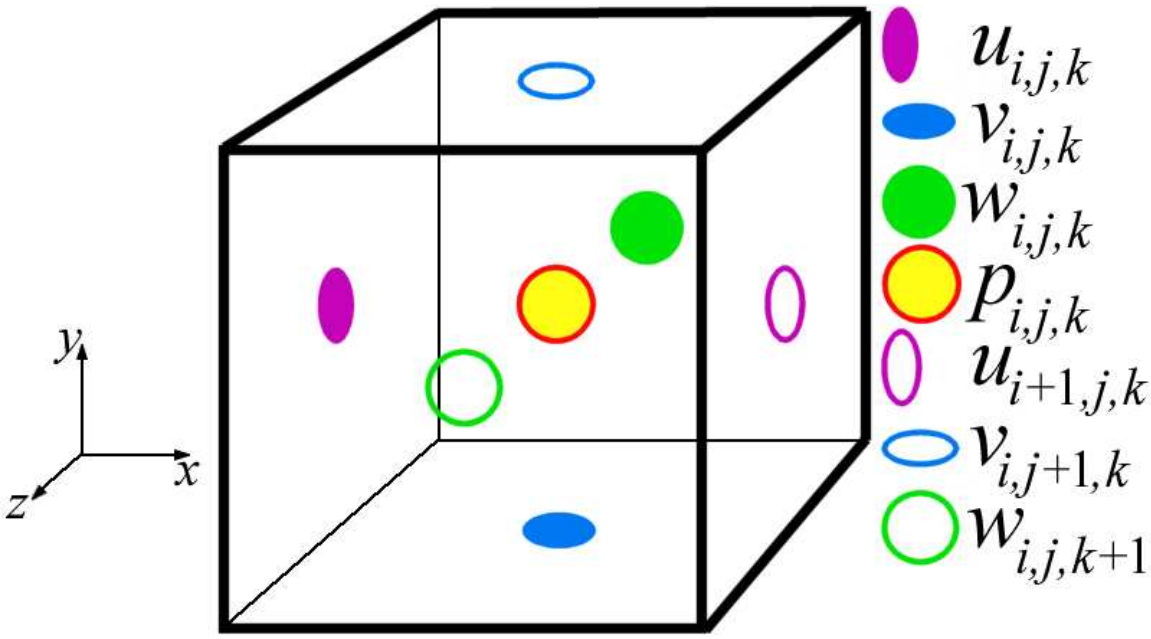

Figure 5: A cell in the MAC grid.

The *cells* of the simulation space are uniformly sized cubes, or voxels, with sides of length Δx. Two types of variables can be stored in one cell–a scalar value, or a vector value. All scalar values are stored at the cell centers, sometimes referred to to as the *cell node*. All vector values are stored in a *staggered grid* formation, in which the x component of the vector is stored at the left face of the cube, the y component is stored at the bottom face of the cube, and the z component is stored at the back face of the cube. These face values are sometimes referred to as *face nodes*. The fluid simulator described in this chapter records two variables in each cell: the pressure and the velocity. The scalar pressure p is calculated and stored at the cell node, and the vector valued velocity $\mathbf{u}=(u,v,w)$ is stored as separate components at the appropriate face nodes. A single cell at location i,j,k is depicted in figure 5. It should be noted that many of the other sources that describe a MAC grid use a *half index notation*. If one wishes to translate the notation used in this thesis to

the half index notation then simply subtract $1/2$ from the appropriate face node locations (*e.g.*, the x component of velocity $u_{i,j,k}$ in our notation, is denoted $u_{i-1/2,j,k}$ in half index notation).

Often, values are needed for variables at locations other than the cell faces or nodes; trilinear interpolation is used to get such needed values. The node of the cell located at the MAC grid location i,j,k is located at $(i\Delta x, j\Delta x, k\Delta x)$ in simulation space. Notation is important when considering interpolated values verses values at a MAC grid location. The value of a variable α at an *interpolated* location, (x,y,z) is denoted $\alpha(x,y,z)$. One specific example of this is the cell-centered velocity vector,

$$\mathbf{u}(i\Delta x, j\Delta x, k\Delta x) = \left(\frac{u_{i,j,k}+u_{i+1,j,k}}{2}, \frac{v_{i,j,k}+v_{i,j+1,k}}{2}, \frac{w_{i,j,k}+w_{i,j,k+1}}{2} \right), \tag{6}$$

where each vector component is stored at the same location. This is different than the velocity stored at the MAC grid location i,j,k, which is

$$\mathbf{u}_{i,j,k} = (u_{i,j,k}, v_{i,j,k}, w_{i,j,k}), \tag{7}$$

and whose components are stored at the three different flux locations $(i\Delta x - \Delta x/2, j\Delta x, k\Delta x)$, $(i\Delta x, j\Delta x - \Delta x/2, k\Delta x)$ and $(i\Delta x, j\Delta x, k\Delta x - \Delta x/2)$.

The computational grid that we use has dimensions $\mathtt{I}\times\mathtt{J}\times\mathtt{K}$, and the grid is indexed with the integers $i \in [0, \mathtt{I}-1]$, $j \in [0, \mathtt{J}-1]$, and $k \in [0, \mathtt{K}-1]$. As mentioned above, the cells on the outside edge of the domain are boundary cells, so fluid can only exist in cells where $1 \le i \le \mathtt{I}-2$, $1 \le j \le \mathtt{J}-2$, and $1 \le k \le \mathtt{K}-2$. The distance scale of the simulation is determined by a *width* in meters, and Δx=*width*/$\mathtt{I}$.

3.2.2 Finite Differences

Finite differences ([68], section 5.7) are used to solve the Navier-Stokes equations on the MAC grid.

The discrete finite difference version of the gradient operator, equation 3, for the pressure at cell i,j,k is a vector, and each of the components are solved for at the appropriate face node,

$$(\nabla p)_{i,j,k} = \left(\frac{p_{i,j,k}-p_{i-1,j,k}}{\Delta x}, \frac{p_{i,j,k}-p_{i,j-1,k}}{\Delta x}, \frac{p_{i,j,k}-p_{i,j,k-1}}{\Delta x} \right). \tag{8}$$

The finite difference version of the divergence, equation 4, of the velocity at cell i,j,k is the scalar

$$(\nabla \cdot \mathbf{u})_{i,j,k} = \frac{u_{i+1,j,k}-u_{i,j,k}+v_{i,j+1,k}-v_{i,j,k}+w_{i,j,k+1}-w_{i,j,k}}{\Delta x}, \tag{9}$$

and is stored at the cell node.

The above finite difference divergence and gradient operators are *first order accurate*. Finite differences uses the Taylor series to approximate derivatives ([93], chapter 1; [18], section 3.2.4). The accuracy of the approximations is measured in the number of terms that the Taylor series is expanded to. First order accuracy means that only the first term is used. If the Taylor series is expanded to the second term, then the approximation would be second order accurate. The finite difference version of the Laplacian operator, equation 5, is second order accurate. Applied to $u_{i,j,k}$, the discrete Laplacian operator is

$$\nabla^2 u_{i,j,k} = \frac{u_{i-1,j,k} + u_{i,j-1,k} + u_{i,j,k-1} - 6u_{i,j,k} + u_{i+1,j,k} + u_{i,j+1,k} + u_{i,j,k+1}}{\Delta x^2}. \tag{10}$$

So far the discussion has been focused on spatial derivatives, but a temporal derivative is needed to approximate $\mathbf{u}_t$. Consider the simple linear equation:

$$\mathbf{u}_t = \lambda \mathbf{u}, \tag{11}$$

where $\lambda < 0$ is some constant. The exact solution to the above equation, $e^{\lambda t}$, decays. The *forward Euler* version of this linear equation is:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \lambda \mathbf{u}^n. \tag{12}$$

The superscript notation is used to represent time. The current time is denoted by n, so the value of $\mathbf{u}$ at the current time step is $\mathbf{u}^n$. A time step of size Δt will take you from time step n to the next time step $n+1$. Forward Euler is an *explicit* approximation of the time derivative, which is to say that the right hand term of equation 12 is already known. Explicit approximations are easy to solve because of this, but they also force a *stability* criteria on the equation that restricts the size of Δt.

Stability is an important area of numerical study, but to go into depth in the subject would take us too far away from the focus of this chapter, which is to inform the reader enough to make them dangerous in the exciting field of computational fluid dynamics. One of the first steps in becoming less dangerous would be to understand stability. Chapter one of Trefethen's book [92] would be a good place for the interested reader to start. This section, however, is an attempt to give the reader an intuitive feel for what stability is without resorting to a full von Neumann stability analysis ([68],

section 19.1). Reconsider equation 12 in a new arrangement,

$$\mathbf{u}^{n+1} = \mathbf{u}^n(1+\Delta t\lambda). \tag{13}$$

Now consider a term I will called the *pseudo-Gain*, $\tilde{\mathbf{G}}$, which is used to analyze a time derivative:

$$\tilde{\mathbf{G}} = \frac{\mathbf{u}^{n+1}}{\mathbf{u}^n}. \tag{14}$$

The idea is to keep $|\tilde{\mathbf{G}}|$ small, but when it is applied to equation 13 it yields

$$\tilde{\mathbf{G}}^{FE} = 1+\Delta t\lambda, \tag{15}$$

so the pseudo-Gain gets larger as $|\lambda|$ and Δt get larger. So, to remain stable with the forward Euler formulation the time step must get smaller as λ gets larger:

$$\Delta t \approx \frac{1}{|\lambda|}. \tag{16}$$

One way to avoid this stability problem is to use an *implicit* formulation like *backward Euler* for the time derivative:

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} = \lambda\mathbf{u}^{n+1}. \tag{17}$$

Rearranging this yields the pseudo-Gain:

$$\tilde{\mathbf{G}}^{BE} = \frac{1}{1-\Delta t\lambda}, \tag{18}$$

and no matter how large $|\lambda|\Delta t$ gets, $|\tilde{\mathbf{G}}|$ stays small. How λ relates to the Navier-Stokes equations, and issues concerning implicit finite difference formulations will be covered in more detail in chapter 4. This chapter will focus on the simpler explicit formulations, which can indeed go a long way even with their shortcomings. Being aware of those shortcomings is good enough for now.

3.2.3 The MAC Method

The Marker-And-Cell method has two major components: the cells, described in section 3.2.1, in which fluid velocity and pressure are tracked, and a large collection of *marker particles* in the fluid that mark which cells are filled with fluid and that carry velocity to previously empty cells.

One time-step in a fluid simulation is calculated in several stages, which will be outlined now and then later described in further detail:

1. An appropriate time step size Δt is chosen.
2. The particles are moved according to the current velocity field, $\mathbf{u}$.
3. Each cell is marked as being fluid-filled or empty according to whether a given cell contains particles, and velocities are set for previously empty cells that now contain fluid.
4. Boundary conditions are set for all solid obstacles and cell faces that are empty on one side and have fluid on the other.
5. The velocity values in the fluid-filled cells are updated based on Equations 1 and 2.

The above steps are repeated for each time-step of the simulation.

3.2.3.1 Finding Δt

The first step is to find an appropriate Δt. Information can only travel so fast across the MAC grid. For the solution to be *consistent* with the true solution a restriction known as the Courant-Friedrichs-Lewy (CFL) condition must be enforced. The CFL condition states that the time step must be small enough to make sure information does not travel across more than one cell at a time. The CFL condition is

$$\Delta t < \frac{\Delta x}{\max(|u|,|v|,|w|)}, \tag{19}$$

and is the first restriction we put on Δt. The other restriction on Δt is imposed by the constant viscosity diffusion equation, $\nu\nabla^2\mathbf{u}$. The variable viscosity version of this equation will be considered in chapter 4. This stability restriction is

$$\Delta t < \frac{\Delta x^2}{6\nu} \tag{20}$$

in 3D. In this stability restriction, ν plays the same role as λ does in equation 16.

3.2.3.2 Moving Particles

After choosing a time step, the particles are moved forward with an explicit second order Runge-Kutta technique known as the *modified Euler* or *Midpoint* method. The particle starts at location $\mathbf{x} = (x,y,z)$ and is updated in two steps with

$$\begin{aligned} \mathbf{x}^{n+1/2} &= \mathbf{x}^n + \Delta t\mathbf{u}(\mathbf{x}^n) \\ \mathbf{x}^{n+1} &= \mathbf{x}^n + \tfrac{\Delta t}{2}\left[\mathbf{u}(\mathbf{x}^n) + \mathbf{u}(\mathbf{x}^{n+1/2})\right]. \end{aligned} \tag{21}$$

If the location $\mathbf{x}^{n+1/2}$ is inside an empty cell, then the second step is not taken because reliable velocities are not available outside the fluid cells. A second order technique is used because it is the highest order that can be achieved when using linear interpolation to get the velocity values.

3.2.3.3 Identifying fluid and empty cells

Once the marker particle positions are updated, they are used to identify cells that contain fluid. If a cell that was once fluid contains no more particles it becomes an empty cell. If a cell was empty, but now contains at least one particle, it becomes a fluid cell. When this happens, velocities must be set for the previously empty cell. Each marker particle has a velocity associated with it, and when a new fluid cell is found, the velocities at the cell faces are obtained by collecting all the particles in a cube centered at the face node and with sides of length Δx. If there are no particles in the area, then the velocity of the nearest particle is used. The positions of the particles near the surface give a highly resolved shape to the free surface, much more detailed than the cells alone. This allows the MAC method to create detailed fluid surfaces while using a relatively coarse cell grid. The fact that the particles save the high frequency detail of the surface is an important point and is discussed further in chapter 6.

3.2.3.4 Boundary Slip and Continuity Conditions

Cell faces that have fluid on each side are discussed in section 3.2.3.5. This section covers the treatment of all cell faces that that have fluid on one side and either solid walls or empty air on the other.

As in [25], we allow any of the cells of the MAC grid to be obstacles that fluid will not enter. In particular, the six sides of the simulation grid are treated as solid walls. We will discuss how this layer of solid boundary cells is treated. The other solid boundary cells are treated similarly. Consider four solid cells on the left side of the MAC grid with fluid cells to the right of these solid boundaries. The four cells are depicted in three ways in Figure 6. Remember the velocity, **u**, has three components, *u*, *v*, and *w*.

The left depiction, in figure 6, highlights in purple the face nodes that hold the *u* components. Like all face nodes that have solid boundaries on one side and fluid cells on the other, these purple values are set to zero to keep fluid from entering the solid boundary.

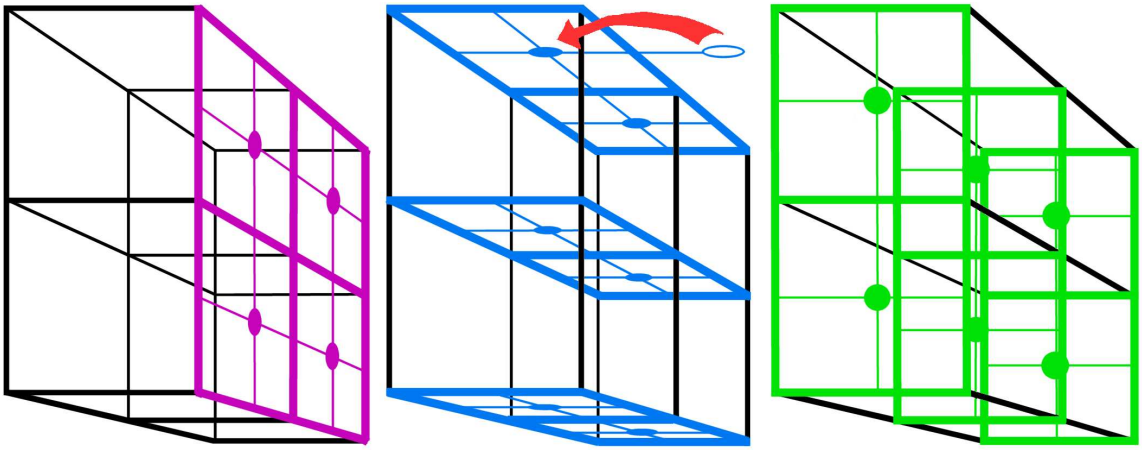

Figure 6: Solid boundary cells on the left side of the MAC grid.

The center depiction, in figure 6, highlights in blue the face nodes that hold the *v* components, and the right depiction highlights in green the face nodes that hold the *w* component. These face nodes hold the tangential velocity of the wall. All these face nodes have solid boundaries on both sides of them. There are three ways to set the values for such nodes. The first way is simply to set the value to zero. The other two ways are known as *slip conditions*; consider the red arrow pointing from a face node with fluid on each side (depicted as the outlined blue oval) to the face node with a solid boundary on each side. Simply copying that value over will create a *free-slip* boundary condition where water splashing near a wall will slide freely along the wall without slowing down. Copying the negative of that value will create what is called a *no-slip* boundary condition, where the tangential velocity at the boundary face where the fluid meets the solid is zero. As Δx approaches zero, the no-slip condition is the more accurate of the slip conditions, but the free-slip condition will keep the animation lively. We allow the animator to choose between free-slip and no-slip conditions for the velocity tangential to these cells.

So far, we have discussed how to set velocities for face nodes that have a solid on one or both sides. Later we will discuss how to solve for face nodes that have fluid on both sides, known as *fluid-faces*, with the Navier-Stokes equations, but first we must describe how to set the value of face nodes that are empty (have air) on one side and filled with fluid on the other–known as *surface-faces*. A cell with at least one surface-face will be called a *surface-cell*. In the following discussion, a *non-surface-face* is either a fluid-face or a cell face with a solid obstacle on either side of it. In

other words, a non-surface-face is a face that we know the value for from boundary conditions or solving the Navier-Stokes equations.

At the surface-faces we do not use slip conditions–a different kind of condition must be satisfied, that of keeping the velocity inside the cell divergence-free. Each surface-cell may have anywhere from one to six faces that are adjacent to air, and each case is treated in order to maintain the divergence-free property. These cases are enumerated in [25, 35] for two dimensions and their extension to 3D is discussed next.

To set the surface-face boundary conditions in a surface-cell we use the continuity condition expressed by equation 1. Throughout the following discussion we will consider a surface-cell at location i, j, k. The discrete version of the continuity condition, using equation 9, is

$$\nabla \cdot \mathbf{u} = 0 \quad \longrightarrow$$

$$u_{i+1,j,k} - u_{i,j,k} + v_{i,j+1,k} - v_{i,j,k} + w_{i,j,k+1} - w_{i,j,k} = 0, \tag{22}$$

which includes the velocity values from all six faces of the cell. If there is only one surface-face on the cell, then the value from the face is set directly with equation 22. For example, if the right face of the cell is the only surface-face, then its value is set with:

$$u_{i+1,j,k} = u_{i,j,k} - v_{i,j+1,k} + v_{i,j,k} - w_{i,j,k+1} + w_{i,j,k}. \tag{23}$$

When there is more than one surface-face on a cell, then there is not a unique solution to the continuity equation. Ideally, equations for the disappearance of tangential stress should be used as extra equations for the other unknowns of the system, but those equations are not known with the MAC method described here. In two dimensions, Harlow noted that enforcing the weaker conditions $u_{i+1,j,k} = u_{i,j,k}$ and $v_{i,j+1,k} = v_{i,j,k}$ result in negligible tangential stresses even for very viscous flows ([99] p.26). When deciding how to set the surface-face values I tried to keep the spirt of Harlow's weaker restrictions in mind, while still ensuring that equation 22 holds true.

There are sixty-three possible surface-cell configurations. Equation 23, and five similar equations, take care of the six cases where there is only a single surface-face. If there are six surface-faces, then nothing is done, the fluid in such a cell will be affected only by the body forces. What follows is a discussion of the other fifty-six cases organized by the number of surface-faces on the

cell. There are many symmetries among these cases so don't worry, a highly detailed discussion of each case is not necessary.

When only two surface-faces exist on one surface-cell there are two configurations that must be considered, one configuration is when air is on two opposite sides of the cell, in this case we do nothing. This can cause divergences, but does seem to allow drips and thin films of fluid particles to fall naturally. In the other case we copy velocities to surface-faces from the faces on the opposite side, and add half the difference of the remaining two faces to each surface-face (another fifteen down, forty-one more to go).

With three surface-faces on a cell, again, there are two configurations that must be considered. In the configuration where three non-surface-face are across from thee surface-face we simple copy values from non-surface-faces to surface-faces. In the other configuration there is one non-surface-face across form a surface face, and also two surface-faces across form each other. Here we solve the velocity for the surface-face which is opposite a non-surface face the same way we do for the single air face case, ignoring the fact that two of the other faces are by air (another twenty down, twenty-one more to go).

Not surprisingly, there are two possible configurations when a cell has four surface-faces. The first is when two surface-faces are opposite one another, but the other two surface-faces are not. Here we copy opposite values to the surface-faces without opposite surface-faces, and again split the divergence of the other two faces and add them (or subtract depending on the arrangement) to get a divergence-free cell. When the four surface-faces each have an opposite surface-face, we calculate the divergence of the cell and add or subtract a quarter of that divergence equally to each surface-face. That takes care of another fifteen configurations, there are only six more to go.

The last six configurations are contained in the case where a cell has five surface-faces. We treat this case the same as the one surface-face case, and the velocity we solve for is the one on the surface-face with an opposite non-surface-face. We note that there is almost no information to work with in this situation because we have one equation and five unknowns. The decision we made with this case may cause noise in the formes of bumps in your fluid surface as discussed in chapter 6.

When interpolating velocity values near the free surface (*e.g.*, when the velocity of a marker particle is needed), sometimes values that are outside the fluid are needed. One good way to get

these velocities is to use an *extension velocity* to extrapolate the velocity along the normal of the surface (see section 6.3). However, we do not have normal information at the surface available at this point, so we simply copy the known velocity component value nearest to the particle.

3.2.3.5 Solving the Navier-Stokes equations

There are two major steps to solving for $\mathbf{u}_t$ with equation 2 while enforcing equation 1, the incompressible fluid constraint. One difficulty in solving equation 2 is that there is no information on how to solve for the pressure p. Luckily, equation 1 actually gives enough information to solve for a suitable pressure. How the pressure is used and how we solve for it will be discussed in section 3.2.4. For now the discussion will ignore the pressure and describe the forward Euler solution to a *best guess velocity*, $\tilde{\mathbf{u}}$.

The equation for the best guess velocity with a constant viscosity is:

$$\tilde{\mathbf{u}} = \mathbf{u} \; + \; \Delta t[\; -(\mathbf{u}\cdot\nabla)\mathbf{u} \; + \; \nu\nabla^2\mathbf{u} \; + \; \mathbf{f} \;], \tag{24}$$

without taking into account the pressure. The finite difference version of this equation is solved one dimension at a time with the following three equations:

$$\begin{aligned}
\tilde{u}_{i,j,k} = u_{i,j,k} + \tfrac{\Delta t}{4\Delta x}[&(u_{i-1,j,k}+u_{i,j,k})(u_{i-1,j,k}+u_{i,j,k}) - (u_{i,j,k}+u_{i+1,j,k})(u_{i,j,k}+u_{i+1,j,k}) \\
&+(u_{i,j-1,k}+u_{i,j,k})(v_{i-1,j,k}+v_{i,j,k}) - (u_{i,j,k}+u_{i,j+1,k})(v_{i-1,j+1,k}+v_{i,j+1,k}) \\
&+(u_{i,j,k-1}+u_{i,j,k})(w_{i-1,j,k}+w_{i,j,k}) - (u_{i,j,k}+u_{i,j,k+1})(w_{i-1,j,k+1}+w_{i,j,k+1})] \\
&+\tfrac{\nu\Delta t}{\Delta x^2}[u_{i-1,j,k}+u_{i,j-1,k}+u_{i,j,k-1} - 6u_{i,j,k} + u_{i+1,j,k}+u_{i,j+1,k}+u_{i,j,k+1}] + \Delta t f,
\end{aligned} \tag{25}$$

$$\begin{aligned}
\tilde{v}_{i,j,k} = v_{i,j,k} + \tfrac{\Delta t}{4\Delta x}[&(v_{i-1,j,k}+v_{i,j,k})(u_{i,j-1,k}+u_{i,j,k}) - (v_{i,j,k}+v_{i+1,j,k})(u_{i+1,j-1,k}+u_{i+1,j,k}) \\
&+(v_{i,j-1,k}+v_{i,j,k})(v_{i,j-1,k}+v_{i,j,k}) - (v_{i,j,k}+v_{i,j+1,k})(v_{i,j,k}+v_{i,j+1,k}) \\
&+(v_{i,j,k-1}+v_{i,j,k})(w_{i,j-1,k}+w_{i,j,k}) - (v_{i,j,k}+v_{i,j,k+1})(w_{i,j-1,k+1}+w_{i,j,k+1})] \\
&+\tfrac{\nu\Delta t}{\Delta x^2}[v_{i-1,j,k}+v_{i,j-1,k}+v_{i,j,k-1} - 6v_{i,j,k} + v_{i+1,j,k}+v_{i,j+1,k}+v_{i,j,k+1}] + \Delta t g, \quad \text{and}
\end{aligned} \tag{26}$$

$$\begin{aligned}
\tilde{w}_{i,j,k} = w_{i,j,k} + \tfrac{\Delta t}{4\Delta x}[&(w_{i-1,j,k}+w_{i,j,k})(u_{i,j,k-1}+u_{i,j,k}) - (w_{i,j,k}+w_{i+1,j,k})(u_{i+1,j,k-1}+u_{i+1,j,k}) \\
&+(w_{i,j-1,k}+w_{i,j,k})(v_{i,j,k-1}+v_{i,j,k}) - (w_{i,j,k}+w_{i,j+1,k})(v_{i,j+1,k-1}+v_{i,j+1,k}) \\
&+(w_{i,j,k-1}+w_{i,j,k})(w_{i,j,k-1}+w_{i,j,k}) - (w_{i,j,k}+w_{i,j,k+1})(w_{i,j,k}+w_{i,j,k+1})] \\
&+\tfrac{\nu\Delta t}{\Delta x^2}[w_{i-1,j,k}+w_{i,j-1,k}+w_{i,j,k-1} - 6w_{i,j,k} + w_{i+1,j,k}+w_{i,j+1,k}+w_{i,j,k+1}] + \Delta t h,
\end{aligned} \tag{27}$$

where the components of the current velocity $\mathbf{u}$ are (u, v, w), the components of best guess velocity $\tilde{\mathbf{u}}$ are $(\tilde{u}, \tilde{v}, \tilde{w})$, and the components of body force $\mathbf{f}$ are (f, g, h). Solving the above three equations on every face cell in the MAC grid with fluid on both sides gives the best guess velocity, $\tilde{\mathbf{u}}$. However, there is no pressure term in $\tilde{\mathbf{u}}$, and equation 1 has not been considered for any cell that is not a surface-cell.

3.2.4 Pressure Projection

We did not consider the pressure immediately because the best guess velocity, $\tilde{\mathbf{u}}$ from the previous section is not divergence-free (*i.e.*, $\nabla \cdot \tilde{\mathbf{u}} \neq 0$), and the next step we must take is to use a solution for the pressure to make the new velocity divergence-free, thus enforcing the incompressibility constraint. This step is known as the *pressure projection* step, and was originally used by Chorin [10], and more information about is can be found in [65], section 6.3. The term in equation 2 that we left out of equation 24 was

$$-\frac{1}{\rho}\nabla p, \tag{28}$$

and we must account for it in the final velocity,

$$\mathbf{u}^{new} = \tilde{\mathbf{u}} \; - \; \frac{\Delta t}{\rho}\nabla p. \tag{29}$$

We also need the final velocity to be incompressible, so we take the divergence of equation 29 to get

$$\nabla \cdot \mathbf{u}^{new} = \nabla \cdot \tilde{\mathbf{u}} \; - \; \frac{\Delta t}{\rho}\nabla \cdot (\nabla p) = 0. \tag{30}$$

Rearranging equation 30 gives us the equation

$$\Delta t \nabla^2 p = \rho \nabla \cdot \tilde{\mathbf{u}} \tag{31}$$

with which we must solve for p. We then substitute p back into equation 29 to complete the pressure projection, thus enforcing the incompressibility constraint.

3.3 Solving a System of Linear Equations

Equation 24 may look ugly when expressed as the finite difference equations 25-27, but everything on the right hand side of those equations is know, and the solutions is direct. Equation 31 is different;

even though the right had side is know, the left hand side contains the Laplacian operator, ∇^2. The Laplacian operator can be represented by equation 10, so for each of the `N` fluid cells in the MAC grid there is one unknown pressure variable, and one equation for that unknown. The equations for the pressure are linear, and the `N` linear equations together with the and `N` unknowns form what is commonly called a *linear system*. Linear systems can be represented in matrix form, and *linear algebra* is used to work with these systems [33].

3.3.1 Setting Up The Matrices

In this section we will describe how to build the linear system in matrix form that will solve equation 31. The notation we will use for this matrix equation is

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{32}$$

In a linear system with `N` equations and `N` unknown values: $\mathbf{A}$ is an $\mathtt{N} \times \mathtt{N}$ matrix where each row represents an equation; $\mathbf{x}$ is a column vector of size `N` with an entry for each of the unknowns, and $\mathbf{b}$ is a column vector of size `N` that represents the constant known values that can be moved over to the right of the equation. For this discussion `N` is the number of fluid-filled cells. A pressure value must be found for each of these cells. To recast equation 31 as the linear system in equation 32, let

$$\mathbf{A} \equiv -\Delta x^2 \nabla^2 \tag{33}$$

be the negative of the Laplacian operator, scaled by the negative of the grid width squared. The negative is used to make the system positive definite instead of negative definite ([33], section 4.2), and we multiply through by Δx^2 to make the matrix entries integer values thus simplifying our discussion. Let

$$\mathbf{x} \equiv p \tag{34}$$

be the unknown pressure values we want to find, and let

$$\mathbf{b} \equiv \frac{\Delta x^2 \rho \nabla \cdot \tilde{\mathbf{u}}}{\Delta t} \tag{35}$$

be the divergence of each fluid cell scaled again by Δx^2, the fluid density, and the inverse of the time step. Remember the discrete divergence we use is equation 9 so the Δx in the denominator cancels one of the ones in equation 35.

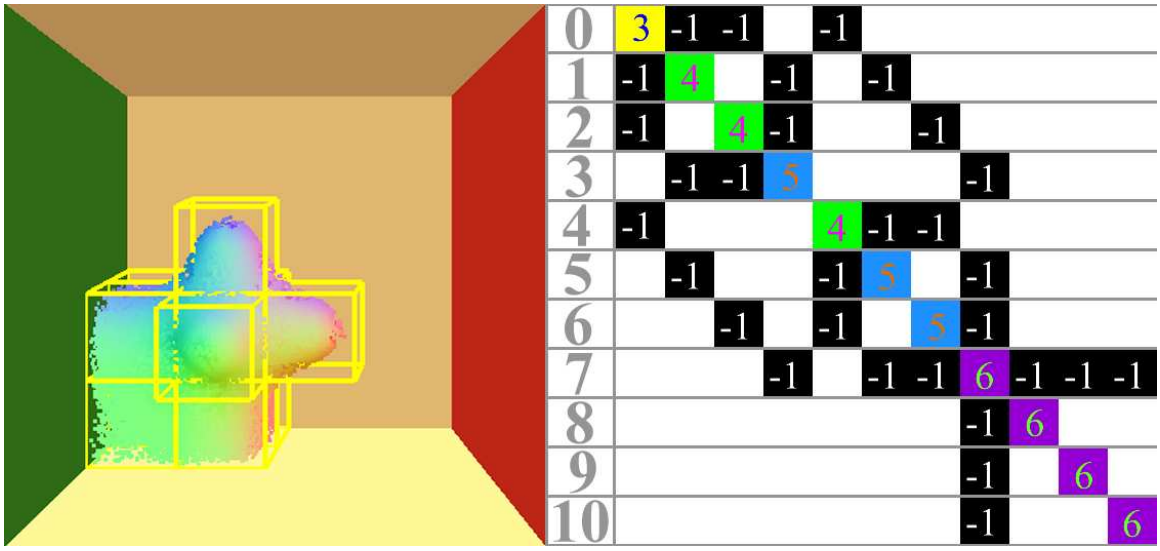

Figure 7: Fluid cells in a MAC grid (left), and the corresponding pressure projection matrix **A** (right), and the indices into the array of fluid cells (center). The white matrix entries $= 0$.

The known constants, **b**, in equation 35 are straightforward to solve for because all the velocity values have been set with the best guess velocity, equation 24, and the continuity condition, equation 22, at the surface-cells. With that knowledge the solution to the right hand side of equation 32 is trivial. Next we will discuss the left hand side, which is much more interesting.

The finite difference form of the left hand side for a fluid-filled cell that has fluid on all sides is similar to equation 10, but the pressure value at the cell center is used instead of the x-component of velocity at the cell face. The next equation transitions from the finite difference version of the left hand side, to the matrix row entry corresponding to that cell:

$$\begin{aligned} -\Delta x^2 \nabla^2 p_{i,j,k} &= -\Delta x^2 \tfrac{p_{i-1,j,k}+p_{i,j-1,k}+p_{i,j,k-1}-6p_{i,j,k}+p_{i+1,j,k}+p_{i,j+1,k}+p_{i,j,k+1}}{\Delta x^2} \\ &= -p_{i-1,j,k} - p_{i,j-1,k} - p_{i,j,k-1} + 6p_{i,j,k} - p_{i,j,k+1} - p_{i,j+1,k} - p_{i+1,j,k} \\ &= (-1 \quad \cdots \quad -1 \quad \cdots \quad -1 \quad 6 \quad -1 \quad \cdots \quad -1 \quad \cdots \quad -1) p_{i,j,k} \\ &= (\mathbf{Ax})_{i,j,k}. \end{aligned} \tag{36}$$

Throughout the following discussion we will be referring to variations of figure 7. Figure 7 depicts the fluid cell configuration in a single simulation frame. The yellow outlined cells are the fluid cells, and the fluid particles are depicted as points of various colors. There are 11 fluid cells in the figure, and the **A** matrix formed by those 11 cells is pictured in the right side of the figure.

There is only one MAC grid cell entry in the depicted simulation frame that is a fluid cell, and

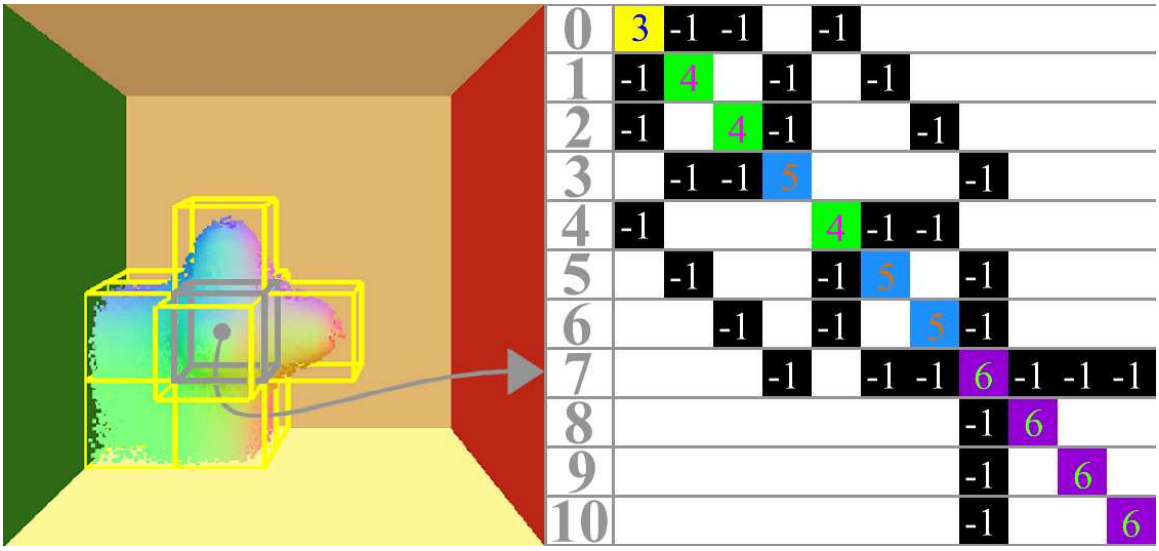

Figure 8: Figure 7 with the only fluid filled cell surrounded by fluid filled cells highlighted in grey.

has a fluid cell on all six sides. This cell corresponds equation 36 with $i, j, k \equiv 2, 2, 2$. This cell is highlighted with grey in figure 8.

When a fluid cell is completely submerged (surrounded by 6 other fluid cells), then we have the exact matrix equation (equation 36) already for that cell's matrix row because it's six neighbors have entries in the matrix as well. Aside from a full cell, there are two other neighboring cell types that a cell may have: an empty air cell, and a solid obstacle cell. Because solid obstacle and empty air cells do not have an entry in **A**, their presence must be accounted for by *boundary conditions*.

3.3.2 Boundary Conditions

There are two types of boundary conditions that we will discuss: *Dirichlet*[1] boundary conditions are set to a known value, and *Neumann*[2] boundary conditions are based on the value of the derivative between the boundary cell and the cell in the matrix.

3.3.2.1 Neumann boundary condition

For the pressure projection matrix, **A**, a Neumann boundary condition of 0 will be set at all the solid obstacle cells. That is to say, there is no change in pressure between solid obstacles and the fluid

[1]There is some dispute over how to pronounce Dirichlet, but I like to pronounce it \ˈde-ri-ˌshlet\, because of where his surname come from,"Le jeune de Richelet" which means "the young chap from Richelet".

[2]Pronounced \ˈnȯi-man\.

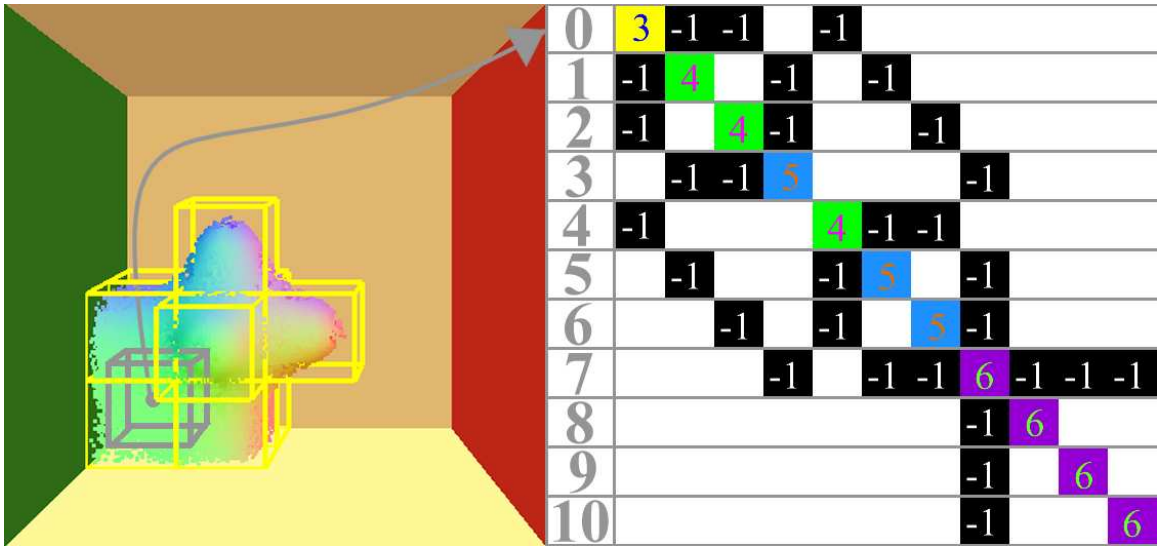

Figure 9: Figure 7 with the back-lower-left fluid filled cell highlighted in grey. This corner cell is special because it has three faces that border solid obstacle cells. Neumann boundary conditions need to be set at these solid obstacles.

cells next to them. This is consistent with our model because the velocities at cell faces between an obstacle and a fluid cell are set to 0 (see section 3.2.3.4), so when the gradient of the pressure is subtracted off the velocity at that face in the final step of the pressure projection, equation 29, the gradient must be zero as well or the pressure will force fluid into or out of the solid walls. In figure 9 the lower left back cell in the simulation grid, at index $i, j, k \equiv 1, 1, 1$, is highlighted in grey. As depicted in the figure, this cell has an array index of 0 in the fluid cell array, and the neighboring cells in front, above, and to the right have array indices of 1, 2, and 4. The three neighboring solid obstacle cells do not have entries in the array, so they will be subscripted with their location relative to the corner cell in this discussion. The matrix equation, second line of equation 36, for the corner cell can be written as:

$$-p_{left} - p_{below} - p_{behind} + 6p_0 - p_1 - p_2 - p_4, \tag{37}$$

and can be rearranged as:

$$(p_0 - p_{left}) + (p_0 - p_{below}) + (p_0 - p_{behind}) + 3p_0 - p_1 - p_2 - p_4. \tag{38}$$

The three differences on the left of equation 38 are the derivatives between the solid walls to the left, below, and behind the corner cell. By setting the Neumann boundary conditions for these cells to 0

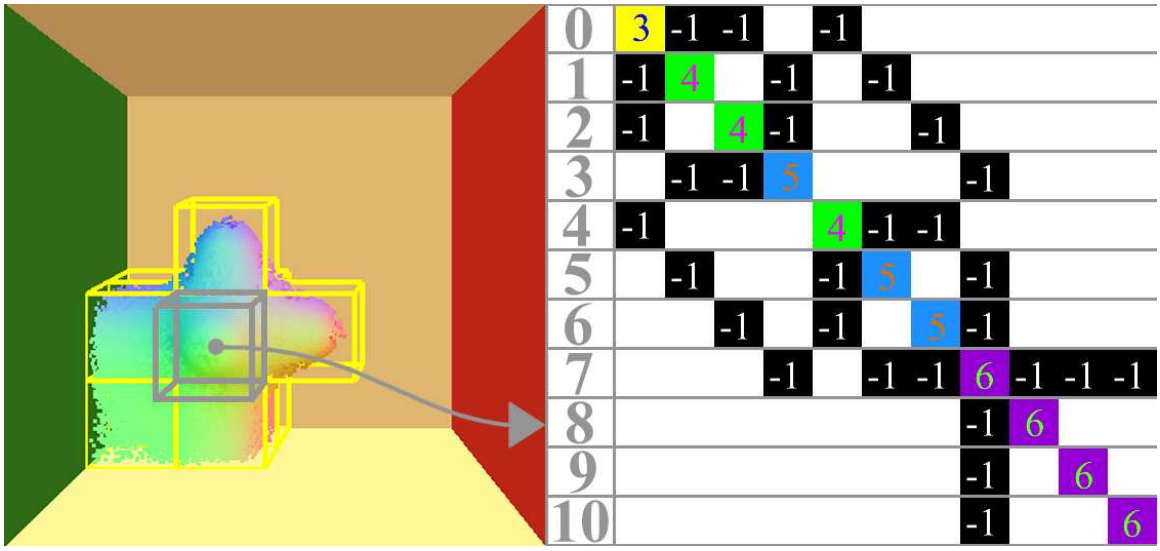

Figure 10: Figure 7 with a fluid filled cell protruding into the air highlighted in grey. This cell is special because it is surrounded by empty air cells on all sides save one. Dirichlet boundary conditions must be set in all the surrounding empty air cells.

we set the three differences to 0 and they drop out of the equation leaving the matrix row pointed to in figure 9. If a Neumann boundary condition other than 0 is needed for some reason, then we could simply move that constant value over to the known vector, **b**, and they would still disappear from the row in **A**.

3.3.2.2 Dirichlet boundary condition

The air pressure in the simulation will be 0, so the pressure in all the empty air cells of the simulation will be set with Dirichlet boundary conditions of zero. Consider the cell highlighted in grey in figure 10; we know that the pressure in the five empty air cells surrounding the cell in the front, the left, the right, above it, and below it must be set to 0. These empty air cells do not have an entry in the fluid cell array. The highlighted cell has an array index of 8, as the figure shows, and the cell behind it has an array index of 7, so the matrix equation for the cell is:

$$-p_{left} - p_{below} - p_7 + 6p_8 - p_{front} - p_{above} - p_{right}. \tag{39}$$

Substituting a value of 0 for the 5 empty air cells in the equation 39 leads to the equation represented by matrix row 8 in figure 10. If an air pressure other than zero is needed, then simply substitute the correct air pressure into equation 39 and then move those known values over to the known vector, **b**.

The fluid cells in the MAC grid can have any combination of Dirichlet or Neumann boundary conditions, but there must be at least one empty air cell represented in the **A** matrix, or the matrix will be singular and can not be inverted uniquely. Another way to think of this is, if there is no air cell, then we can not divide equation 32 through by **A** (*i.e.*, $\mathbf{x} = \mathbf{b}/\mathbf{A}$) to get the correct answer for **x**. In [17] this is done by setting Dirichlet boundary conditions for all cells at the top of the simulation, effectively making the simulation open to the air above.

After setting up **A** with the boundary conditions above, and computing the vector of known values, **b**, we are ready to solve the solution vector, **x**, in our system of linear equations. We will do this, not by dividing through by **A**, but by solving a minimization with a technique known as *conjugate gradient* (CG). Before describing the CG solver, we will side track a bit and talk about the sparse matrix storage structure used to hold **A**.

3.3.3 Sparse Matrix Storage

The pressure projection matrix, **A**, is a *sparse* matrix. That is to say, it contains mostly zeroes. To make our simulations tractable we can not explicitly store all the zero entries. For example, the simulations in figures 20 and 21 were run with a grid size of $68 \times 292 \times 24$ and were almost completely full of fluid. For arguments sake, lets say they are completely filled with fluid so the number of fluid cells is $\mathtt{N} = 476,544$. The floating point values in **A**, **x**, and **b** will all be stored as 8 byte doubles. The 3.6 megabytes (MB=1,048,576 bytes) of storage needed for both **x** and **b** can not be avoided, however the machine used to run those simulations has 2.0 gigabytes (GB=1,024MB) of memory so their footprint is relatively small. Remember, however, that **A** represents an $\mathtt{N} \times \mathtt{N}$ matrix, so if we store a double for every entry in the matrix it would take $1,692.0$ GB of memory. As mentioned above, most of those entries are 0 so if we can ignore them we will be able to fit **A** into memory. Ignoring all boundary conditions, the maximum number of non-zero entries in **A** is $7\mathtt{N}$. This implies that we should be able to create an $\mathrm{O}(\mathtt{N})$ scheme for storing and accessing **A**.

There are many types of sparse matrices ([72], chapter 3), and many different ways to store them. When choosing a storage scheme one should consider what operations need to be done with it (*e.g.*, matrix/matrix multiplication or matrix/vector multiplication), what kind of matrix is to be stored (*e.g.*, structured or unstructured), and ease of programming (*e.g.*, using less storage may need

```
typedef struct {
    byte   numOff        /* the number of off diagonal terms */
    double diag          /* value of the diagonal entry */
    double offEntries[6] /* value of the off diagonal entries */
    int    offIndices[6] /* array index of the off diagonal terms */
} matrix-row
```

Table 1: The `matrix-row` data structure used for the sparse matrix storage of **A**.

complicated control loops).

Each row in **A** represents one equation, so it was natural to base the sparse storage scheme on a row based structure. We use the `matrix-row` structure listed in table 1, which is 81 bytes in size (ignoring aligned memory allocation). So the storage needed for **A** is $O(N)$ when we store one `matrix-row` structure for each of the `N` fluid cells, and for the above example it would take 36.8 MB, which is acceptable and will fit into our 2.0 GB memory limit just fine. Next we will look at each entry in the `matrix-row` structure in more detail, and then discuss alternatives that can be used in its place.

The `numOff` entry has to store the number of off diagonal terms that are in the row, and it is only one byte in size because we know that it only needs to hold values between 0 and 6. The `daig` entry stores the value at the diagonal of the matrix row as an 8 byte floating point value, and `offEntries` is an array that holds the off diagonal values. If there are less than 6 off diagonal terms in the row then the last entries in `offEntries` are not accessed at all, but they still take up memory. This is true for `offIndices` as well and we use this scheme to avoid complicated memory allocations. The `offIndices` holds an integer entry that is the index into the fluid cell array that the off diagonal entries correspond to. The indices are needed because, when multiplying a matrix and a vector, we need to know the index into the vector that the column entry corresponds to (obviously the column entry for `diag` is the same as the row entry because it is on the matrix diagonal).

Using `A`, an `N` sized array of `matrix-row` structures to represent a matrix, as well as `p` and `q`, two `N` sized arrays of floating point values, we can compute and store a matrix-vector multiply with the code in table 2.

There are of course alternatives to using the `matrix-row` data structure. There are ways to exploit the symmetry of **A** that can save storage, but in the special case we are considering there is

```
/* Assume that A, p, and q are allocated and filled correctly.*/
/* A[0] has type matrix-row (table 1).  p[0] and q[0] have type double.*/
for(int n = 0; n < N; n++){
    q[n] = A[n].diag * p[n]
    for(int m = 0; m < A[n].numOff; m++){
        q[n] += A[n].offEntries[m] * p[ A[n].offIndices[m] ]
    }
}
```

Table 2: Pseudo-code to solve a matrix vector multiplication, *i.e.*, $\mathbf{q} = \mathbf{Ap}$.

an even larger space saver that we could use. All the non-zero off diagonal terms in the pressure projection matrix described in this chapter are -1, so we never need to store their value at all because they are known. Furthermore, the diagonal terms are never greater than 6, so only 3 bits are needed to store both the number of off diagonals and the value at the diagonal. Lastly, the offset instead of the index of the fluid cell array can be stored. There are six ways to translate from the MAC grid to the fluid cell index array by running though the different dimensions of the MAC grid in different orders[3]: ijk, ikj, jki, jik, kij, or kji. If the largest dimension (in the above example J$=292$ is the largest) is used first, then the largest offset that can be stored in the fluid cell array is the multiple of the two smallest indices. So, with this scheme, 6 bits are needed for each fluid cell to store the number of off diagonal terms and the off diagonal values, and $\lceil \log_2(\min(\mathtt{IJ}, \mathtt{IK}, \mathtt{JK})) \rceil + 1$ bits are needed for each off diagonal terms. The extra bit is because the offset can be negative. For the example we have been using, this scheme would take 4.4 MB to store all information needed for the pressure projection matrix.

We chose not to describe this scheme in detail here because implementing it is an exercise in bit shifting and non-portable code writing, and also because we use the same `matrix-row` data structure to hold the implicit variable viscosity matrix that is described in chapter 4–the entries in that matrix must be stored as doubles. Now that we have described the sparse matrix structure, we will describe the algorithm it was designed for, the conjugate gradient solver.

[3]Care must be taken here, because accessing the MAC grid in a non-contiguous order can be much slower than contiguous access. Of course this cost is amortized by the fact that it need only be done once per simulation frame to build the fluid cell array, and the fluid cell array will be accessed several times.

3.3.4 Conjugate Gradient Solver

The most naive way to solve the linear system $\mathbf{A}\mathbf{x}=\mathbf{b}$ would be to find $\mathbf{A}^{-1}$ directly and then set $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$. Unfortunatly, the sparse matrix property of $\mathbf{A}$ does not hold for $\mathbf{A}^{-1}$, which in general can be a full $\mathtt{N}\times\mathtt{N}$ matrix. Since $\mathbf{A}^{-1}$ can be a full matrix, the simple act of finding $\mathbf{A}^{-1}\mathbf{b}$ becomes an $\mathrm{O}\left(\mathtt{N}^2\right)$ operation, which in reality we can not carry out reliably anyway, when $\mathtt{N}$ is large, because of numerical error.

One alternative is to use the *conjugate gradient* (CG) method ([33], sections 9.3 & 10.2; [4], section 2.3.1; [72] section 6.7) which minimizes

$$\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}-\mathbf{x}^T\mathbf{b}. \tag{40}$$

Setting $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$ gives the minimum value of equation 40, which is $-\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}/2$. Therefore, minimizing equation 40 and solving for $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$ are equivalent problems.[4]

The CG algorithm is listed in table 3. During each loop of the algorithm, a new *iterate* (stored in $\mathbf{x}$) is chosen. An optimal *search vector* (gradient), $\mathbf{p}$, is chosen such that moving along $\mathbf{p}$ by a distance of α will get us closer to the true solution. Each loop iteration, a search vector is chosen that is orthogonal (conjugate) to all the previous search vectors. As long as $\mathbf{A}$ is symmetric positive definite (which it will always be in the pressure projection matrix as long as there is at least one air cell, see section 3.3.2.2), CG will converge eventually. For most simulations we use $\varepsilon = 10^{-8}$ and $\mathtt{iter}^{max} = 1000$. The initial guess, stored in $\mathbf{x}$, is the previous solution for the pressure.

[4] This is true only when *linearsystem* is symmetric and positive definite.

```
/* N is the length of all the arrays. A is an array of type matrix-row,*/
/* and b is an array of known values.  The x array initially holds a   */
/* guess; each loop x is updated with the new iterate, and at the end  */
/* it holds the solution. The r array holds the residuals, and p and q */
/* store intermediate values. α, β, ρ, ρ^old, and b^norm are scalars.    */
/* iter^max is the maximum number of iterations used to reach the stop- */
/* ping criteria of √ρ <= εb^norm, where ε is the absolute tolerance.  */
ρ = 0.0
b^norm = √(b·b)
r = b - Ax  /* matrix vector multiply is in table 2 */
for(int i = 0; i < iter^max; i++){
    ρ^old = ρ
    ρ = r·r
    if(ρ == 0.0 || √ρ <= εb^norm) break;  /* Check for convergence.  */
    if(i == 0){
        p = r
    } else {
        β = ρ/ρ^old
        p *= β
        p += r
    }
    q = Ap  /* Table 2 */
    α = ρ/(p·q)
    x += αp
    r -= αq
}
```

Table 3: Pseudo-code for the conjugate gradient algorithm.

CHAPTER IV

MELTING AND FLOWING

"IT IS NOT ONCE NOR TWICE BUT TIMES WITHOUT NUMBER THAT THE SAME IDEAS MAKE THEIR APPEARANCE IN THE WORLD."

Aristotle

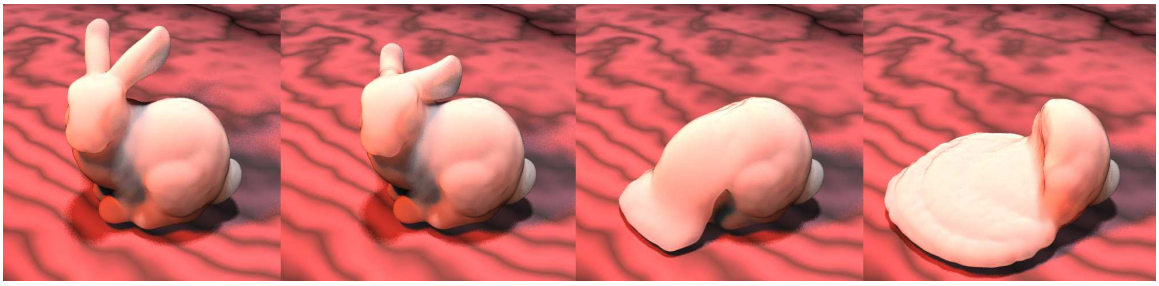

Figure 11: A melting wax bunny.

The idea for the research in this chapter started as a geometric modeling one. The idea was to take a triangle mesh, run some "melting" operation on it, and receive a molten model in return. Early on we decided that having all the frames in-between the original and molten model would be a very cool animation project. With some more research, and a little stubbornness on my part, the melting project became a melting and flowing project that would have a fluid model at its core. The question then became how do we model melting, like that of wax, as a fluid process. There is a linear relationship between the temperature and the viscosity of petroleum wax, so we knew that variable viscosity was necessary, and for simplicity, we decided that even solid wax would be modeled with viscosity, a very high viscosity. The MAC method for simulating liquids was popularized in computer graphics by Foster and Metaxas [25], and it represented the surface/air interface that we knew we would need. The MAC method is a finite difference approach. Chapter 3 describes how to numerically solve the constant viscosity Navier-Stokes equations using the MAC method with explicit forward Euler for the viscous terms. The following sections of this chapter examine how the MAC method may be modified to handle high viscosity and variable viscosity

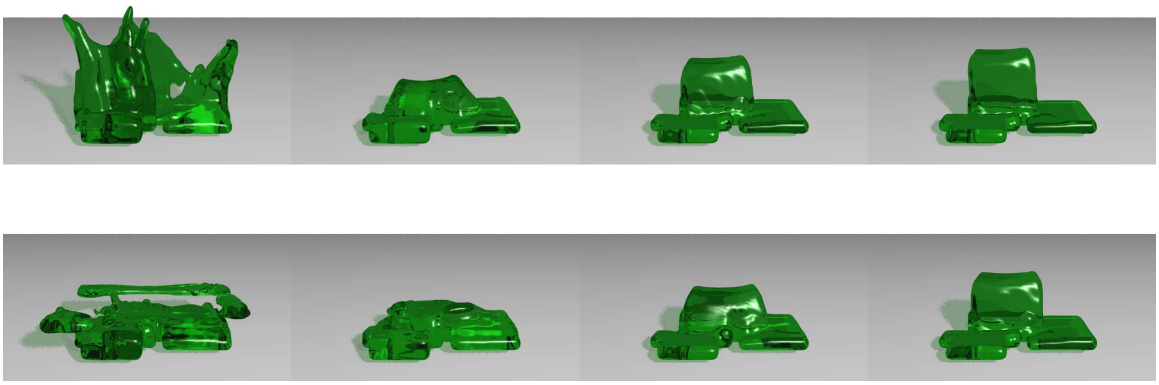

Figure 12: Pouring several different viscous liquids into a container with complex boundaries. Top row is after 1.6 seconds, bottom row is after 10 seconds. From left to right, the fluid viscosities are 0.1, 1, 10, and 100.

fluid. With these modifications we are able to create a simulator that realistically melts materials.

4.1 High Viscosity

The MAC method as described in chapter 3 is well-suited to simulating fluids with relatively low viscosity. This approach has become a favorite for computer graphics because of its ability to capture not only surface ripples and waves but full 3D splashes. Unfortunately, as it stands, the MAC method cannot simulate high viscosity fluids with free surfaces without introducing prohibitively many time steps. In order for the algorithm to remain stable, the method must respect a CFL condition, equation 19, restricting the maximum speed with which information can propagate in one time step from a cell to its neighbors. Additionally, the explicit implementation of the MAC method must also obey a stability criterion, equation 20, imposed to prevent numerical instability in the calculation of the momentum diffusion contribution; at high viscosities, this second stability criterion for explicit solvers becomes more stringent than the CFL condition.

Consider a simplification of equation 2 that only accounts for the momentum diffusion term with constant viscosity:

$$\mathbf{u}_t = \nu \nabla^2 \mathbf{u}. \tag{41}$$

We calculate the new x component of the velocity at cell face $u_{i,j,k}$, after a time-step Δt using central

differencing as follows:

$$u_{i,j,k}^{new} = u_{i,j,k} + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1,j,k} + u_{i,j-1,k} + u_{i,j,k-1} - 6u_{i,j,k} + u_{i+1,j,k} + u_{i,j+1,k} + u_{i,j,k+1}). \quad (42)$$

As mentioned above, this equation assumes that we are dealing with constant viscosity, a restriction that we will relax in section 4.3.

When the viscosity ν becomes large, the viscous diffusion part of the time evolution exhibits *stiffness* ([46], chapter 6). Stiffness is an elusive phenomenon to define; for the purposes of this discussion, stiffness exists when the time step Δt is restricted by a stability criteria rather than accuracy. The finite difference approximation to the viscous contribution, as described for a simple forward Euler step in equation 42, has eigenvalues between $(1 - 4d\nu\Delta t/\Delta x^2)$ and 1, in d dimensions, as indicated by a straightforward von Neumann stability analysis ([68], section 19.1). Thus, to prevent numerical instability, the time step must remain small enough so that $\nu\Delta t/\Delta x^2 < \frac{1}{2d}$, which can become prohibitively small for large viscosity ν. Similarly, since there are no so-called "A-stable" explicit schemes, higher-order explicit time steps (e.g., fourth-order Runge-Kutta) meet with similarly prohibitive stability criteria at large viscosities, at only marginally different threshold values [92]. Lowering the time-step size is one possible fix to this problem, but this quickly leads to a prohibitively large number of time steps: even moderately viscous fluids with a viscosity of 10 require that 6000 time-steps be taken using forward Euler integration to simulate one second of fluid motion. [1] The required time-steps goes up linearly with viscosity; *i.e.*, a viscosity of 100 would require 60,000 time-steps. The approach that we describe below allows fluids with 100 viscosity to be simulated using 30 time-steps per second, so long as the CFL condition is also obeyed.

The solution we propose to the problem of highly viscous fluids requires the replacement the forward Euler integrator for diffusion with an implicit Euler step within the MAC method. This implicit integration is stable at arbitrarily high viscosities, and we give details on how to accomplish this starting with a discussion of *operator splitting*.

[1] For all the viscosities quoted in this chapter a cell size of $\Delta x = 0.1$ is used, and the kinematic viscosity has units of $space^2/time$.

4.1.1 Operator Splitting

To replace the diffusion component of equation 2 with an implicit integration method, we first have to separate the diffusion term from the calculation of advection and body forces. We do this using a standard approach known as operator splitting ([68], section 19.3). The idea of operator splitting is to separate the right-hand components of a PDE (like equation 2) into multiple terms, and to calculate these terms in sequence, independently of one another. Thus, if each individual numerical procedure is stable, the sequence of calculations for successive terms is also stable. In chapter 3 we ignored the pressure term, which is implemented at the end of a velocity update to maintain incompressible divergence-free fluid motion, and found the best guess velocity $\tilde{\mathbf{u}}$ with equation 24. If we perform partial operator splitting, separating out the diffusion term, we get:

$$\tilde{\mathbf{u}} = \mathbf{u} \; + \; \Delta t[\; -(\mathbf{u}\cdot\nabla)\mathbf{u} \; + \; \nu\nabla^2\mathbf{u} \; + \; \mathbf{f} \;] \quad \longrightarrow$$

$$\mathbf{u}^{temp} = \mathbf{u} \; + \; \Delta t[\; -(\mathbf{u}\cdot\nabla)\mathbf{u} \; + \; \mathbf{f} \;] \tag{43}$$

$$\tilde{\mathbf{u}} = \mathbf{u}^{temp} \; + \; \Delta t \nu\nabla^2\mathbf{u}^{temp}. \tag{44}$$

There is only one change between the above two equations and equation 24. The change is that the diffusion of equation 44 is calculated based on an intermediate value $\mathbf{u}^{temp}$ of the velocity instead of the original velocity $\mathbf{u}$. This is an important difference, however, because it allows us to use methods other than forward Euler to calculate the contribution of diffusion. In particular, we use an implicit Euler scheme, which is stable even for high viscosities.

4.1.2 Implicit Viscous Diffusion Formulation

Next we will discuss how to set up a system of equations, based on equation 42, in a matrix formulation. Equation 42 is only for the *x* component of the velocity, similar equations are solved for the *y* and the *z* components. After setting this matrix up, we will describe how to use it to form an implicit backward Euler solution to the viscous diffusion term of the Navier-Stokes equations. The implicit solution will ultimately allow us to simulate fluids with very high viscosity, so high in fact that the fluid will act very much like a solid.

In order to re-formulate the diffusion calculation, here is the central difference diffusion equation (Equation 42) in matrix form:

$$\begin{bmatrix} \vdots \\ u^{new}_{i-1,j,k} \\ \vdots \\ u^{new}_{i,j-1,k} \\ \vdots \\ u^{new}_{i,j,k-1} \\ u^{new}_{i,j,k} \\ u^{new}_{i,j,k+1} \\ \vdots \\ u^{new}_{i,j+1,k} \\ \vdots \\ u^{new}_{i+1,j,k} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ u_{i-1,j,k} \\ \vdots \\ u_{i,j-1,k} \\ \vdots \\ u_{i,j,k-1} \\ u_{i,j,k} \\ u_{i,j,k+1} \\ \vdots \\ u_{i,j+1,k} \\ \vdots \\ u_{i+1,j,k} \\ \vdots \end{bmatrix} + \frac{\nu\Delta t}{\Delta x^2} \begin{bmatrix} \ddots & & \ddots & & \ddots & \ddots & \ddots & & \ddots & & \ddots & & \\ & 1 & \dots & 1 & \dots & 1 & -6 & 1 & \dots & 1 & \dots & 1 & \\ & & \ddots & & \ddots & & \ddots & \ddots & \ddots & & \ddots & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ u_{i-1,j,k} \\ \vdots \\ u_{i,j-1,k} \\ \vdots \\ u_{i,j,k-1} \\ u_{i,j,k} \\ u_{i,j,k+1} \\ \vdots \\ u_{i,j+1,k} \\ \vdots \\ u_{i+1,j,k} \\ \vdots \end{bmatrix} \tag{45}$$

This can be written more compactly in the matrix notation used in section 3.3.1:

$$\boldsymbol{u}^{new} = \boldsymbol{u} + \mathbf{D}\boldsymbol{u} \tag{46}$$

where the term $\boldsymbol{u}$ is a vector that contains the x component of the velocities for each cell face in the MAC grid that has fluid on both sides. The matrix $\mathbf{D}$ is the product of the viscosity constant, the time-step, and the Laplacian operator (Equation 10).

Now that we have an explicit matrix formulation, let us examine the issue of what solver to use. As mentioned earlier, high viscosity fluid would require very small time steps if we use forward Euler integration. The diffusion step can be made stable even with large time steps by reformulating it using implicit backwards Euler integration, though any L-stable ([46], section 6.3) method would be appropriate:

$$\boldsymbol{u}^{new} = \boldsymbol{u} + \mathbf{D}\boldsymbol{u}^{new}. \tag{47}$$

If we define a new implicit diffusion matrix,

$$\mathbf{A} = \mathbf{1} - \mathbf{D}, \tag{48}$$

where $\mathbf{1}$ is the *identity matrix*, then equation 47 can be re-written as:

$$\mathbf{A}\boldsymbol{u}^{new} = \boldsymbol{u}. \tag{49}$$

The implicit diffusion matrix **A**, used in equation 49, is very similar to the pressure projection matrix in equation 32. In order to make use of a simple conjugate gradient solver (section 3.3.4) for equation 49, we require a matrix **A** that is symmetric and positive definite [33]. In creating this matrix, we must also take care to incorporate all of our boundary conditions. Recalling that the vector $\boldsymbol{u}$ contains the velocities at the cell faces, we will describe how we can create such a matrix.

There is a row in **A** for every x component cell face that has fluid in the cells on each side of it. Thus the surface-faces and the faces with a boundary on one or both sides are not represented in the matrix, but they must still be accounted for in the final system of equations. This makes the matrix much smaller then if we were to include every face in the grid. Even though the matrix does not include entries for surface-faces or faces with boundaries, these faces are needed to correctly solve the matrix and are accounted for with Dirichlet boundary conditions (see section 3.3.2 for details on different types of boundary conditions). In order to have the correct velocities at the edges of the simulation, we set boundary conditions after solving equation 43 the same way we do in section 3.2.3.4. After setting our boundary conditions we can hold constant any value that does not have an entry into the diagonal of the matrix. This allows us to move all the known values over to the $\boldsymbol{u}$ vector as Dirichlet boundary conditions, and our matrix stays symmetric.

After incorporating the boundary conditions described above, we arrive at a matrix **A** that is positive definite, symmetric, sparse, and banded. With a large range of viscosities, the condition number of the resulting matrix prevents an effective direct solve, so we solve the equation iteratively using the conjugate gradient method with a Jacobi preconditioner [4]. The pseudo-code for a general preconditioned conjugate gradient is listed in table 4. The Jacobi preconditioner is simply the diagonal of the matrix, so its inverse is the reciprocal of each entry. A Jacobi preconditioner is used because it improves the condition number of the matrix so that it can be solved, it is not used for a fast convergence rate. If a fast convergence rate is needed, then we suggest the incomplete Cholesky preconditioner ([72], section 10.8.2)). The incomplete Cholesky preconditioner is more difficult to implement, and the inversion of the preconditioned is usually an iterative process in itself, so care must be take with the implementation to make the incomplete Cholesky preconditioner worth the effort.

Neither operator splitting nor implicit integration are new to computer graphics. Stam [79] used

```
/* Variables defined in table 3 except the matrix M and the vector z.  */
ρ = 0.0
b^norm = √(b·b)
r = b - Ax  /* matrix vector multiply is in table 2 */
for(int i = 0; i < iter^max; i++){
    solve Mz = r /* M is the preconditioner matrix.  */
    ρ^old = ρ
    ρ = r·z
    if(ρ == 0.0 || √ρ <= εb^norm) break;  /* Check for convergence.  */
    if(i == 0){
        p = z
    } else {
        β = ρ/ρ^old
        p *= β
        p += z
    }
    q = Ap  /* Table 2 */
    α = ρ/(p·q)
    x += αp
    r -= αq
}
```

Table 4: Pseudo-code for the preconditioned conjugate gradient algorithm.

the operator splitting technique so that he could use a semi-Lagrangian method for calculating the advection term of the Navier-Stokes equation, thus making this component of the simulator stable even with very large time-steps. Moreover, Stam uses an implicit Euler integration scheme for calculating diffusion similar to the technique we use. He used an FFT-based solver for diffusion, and thus the particular solver that he used will not be useful for problems with more complex boundary conditions.

4.2 Heat Equation

In order to simulate materials that melt and harden, it is necessary to vary the viscosity according to the properties of the material. In particular, we simulate the temperature changes of the material and we vary the viscosity according to this temperature. Several other graphics researchers have incorporated thermal diffusion and the resulting changes to viscosity into their material models, usually with a particle-based approach [50, 91, 90, 84]. Incorporating these effects into the MAC framework is straightforward, and we give details of how to do so now.

The change in heat is governed by an equation that is very similar to the second part of the

Navier-Stokes equation that we saw earlier. The heat diffusion equation that gives the change in temperature t is:

$$\mathrm{t}_t = k\nabla^2 \mathrm{t} - (\mathbf{u}\cdot\nabla)\mathrm{t}. \tag{50}$$

This equation has two right-hand terms: the diffusion of heat and heat convection. The parameter k is called the *thermal diffusion constant*, and it takes on a small value for those materials that we simulate. The larger k is the faster heat will move through a material. Just as with equation 2, we use operator splitting to solve for changes in temperature. We first use upwind differencing to determine an intermediate temperature due to convection ([68], section 19.1). Then we use an implicit solver that operates on these intermediate values to account for thermal diffusion. We could use the same conjugate gradient solver for heat diffusion as we did for velocity diffusion, but because the thermal diffusion constant k is small we have the luxury of taking a different approach.

To solve for thermal diffusion we perform what is in fact another example of operator splitting with a technique known as the *locally one-dimensional* (LOD) method [52]. We first define a matrix, $\mathbf{H}$, that is a combination of the identity matrix and the one-dimensional Laplacian scaled by the time-step and the thermal diffusion constant (with appropriate boundary conditions[2]):

$$\mathbf{H} = \mathbf{1} - \frac{k\Delta t}{\Delta x^2}\begin{bmatrix} \ddots & \ddots & \ddots & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \end{bmatrix}. \tag{51}$$

Next we arrange the temperature vector t, which contains one value for every center node in each fluid filled cell in the MAC grid, into three different sequences ordered by dimension:

$$^{\mathbf{x}}\mathrm{t} = \begin{bmatrix} \vdots \\ t_{i-1,j,k} \\ t_{i,j,k} \\ t_{i+1,j,k} \\ \vdots \end{bmatrix}, \quad ^{\mathbf{y}}\mathrm{t} = \begin{bmatrix} \vdots \\ t_{i,j-1,k} \\ t_{i,j,k} \\ t_{i,j+1,k} \\ \vdots \end{bmatrix}, \quad \text{and} \quad ^{\mathbf{z}}\mathrm{t} = \begin{bmatrix} \vdots \\ t_{i,j,k-1} \\ t_{i,j,k} \\ t_{i,j,k+1} \\ \vdots \end{bmatrix}. \tag{52}$$

Note that $^{\mathbf{x}}\mathrm{t}$, $^{\mathbf{y}}\mathrm{t}$, and $^{\mathbf{z}}\mathrm{t}$ contain the same temperature values, they are just in a different order. The LOD method solves for t^{n+1} by starting with t^n and going through sequential steps using the three

[2]With the appropriate boundary conditions there will actually be three $\mathbf{H}$ matrices, one for each ordering of the temperature vector.

```
/* The diagonal of A is stored in the N length array, d[].  */
/* The superdiagonal of A is stored in the N-1 length array, e[].  */
/* b[], the N array of known values, is overwritten with the solution.  */
for(int k = 1; k < N; k++){
    t = e[k-1]
    e[k-1] = t/d[k-1]
    d[k] -= t*e[k-1]
}
for(int k = 1; k < N; k++){
    b[k] -= e[k-1]*b[k-1]
}
b[N-1] /= d[N-1]
for(int k = N-2; k >= 0; k--){
    b[k] = b[k]/d[k] - e[k]*b[k+1]
}
```

Table 5: Pseudo-code for a symmetric tridiagonal system solver.

equations:

$$\mathbf{H}^{\mathbf{x}}\mathrm{t}^{n+1/3} = {}^{\mathbf{x}}\mathrm{t}^{n}, \tag{53}$$

$$\mathbf{H}^{\mathbf{y}}\mathrm{t}^{n+2/3} = {}^{\mathbf{y}}\mathrm{t}^{n+1/3}, \text{ and} \tag{54}$$

$$\mathbf{H}^{\mathbf{z}}\mathrm{t}^{n+1} = {}^{\mathbf{z}}\mathrm{t}^{n+2/3}. \tag{55}$$

Each of these three equations is set up to calculate diffusion only in a single direction, either *x*, *y*, or *z*. By solving them in sequence and passing each one's results to the next, we are performing three-way operator splitting. This time, however, instead of splitting one large PDE into separate ones, we are splitting the 3D Laplacian operator into three separate one-dimensional Laplacian operators. This will not yield the same exact answer as the full 3D Laplacian, but it gives a close approximation.

The matrix, **H**, is symmetric, positive-definite, and tridiagonal; solvers for such matrices are fast. We use the LDL^T symmetric tridiagonal system solver described in [33], Section 4.3.6, and depicted in table 5. In particular, doing so is substantially faster than using the preconditioned conjugate gradient solver that we used to solve equation 49. If our thermal diffusion constant *k* had been large, we would have been obliged to use the slower conjugate gradient solver to get accurate results. When we solved for velocity diffusion, the analogous material parameter was the viscosity ν, which can be quite high, so we had to use the computationally more expensive solver.

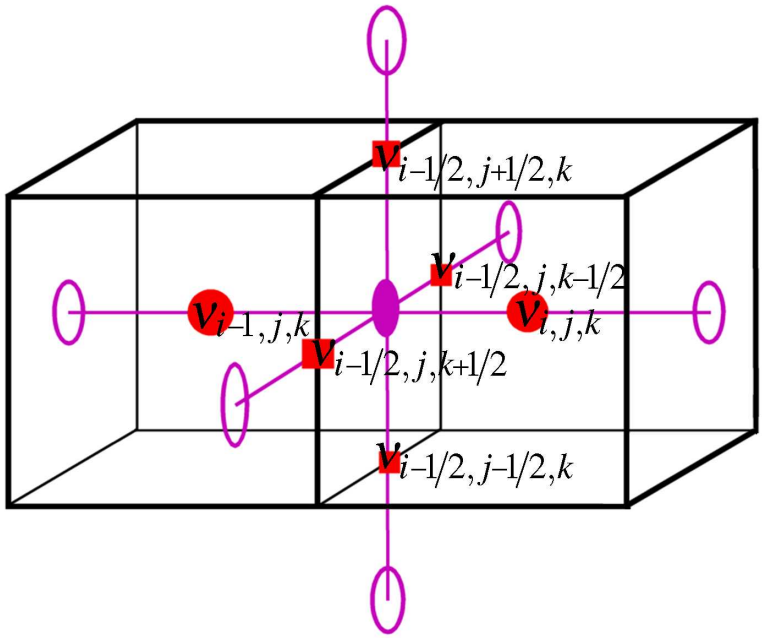

Figure 13: The viscosity values used to calculate viscous diffusion at $u_{i,j,k}$ (the filled purple oval).

Operator splitting by dimensions is a common technique, and some of the more popular such methods are called *alternating-direction implicit* (ADI) methods [68, 52]. The LOD method we use is a closely related technique that is stable in 3D, but ADI techniques that are stable in 3D such as Douglas-Rachford would also be suitable for this task.

In section 3.2.3 we describe the marker particles that carry velocities to newly-filled fluid cells. The particles are also given temperature values, so that they can carry that temperature to the same newly-filled fluid cells.

4.3 Variable Viscous Diffusion[3]

Once we have calculated temperature at each fluid cell center in the simulation, we can use this temperature to determine the material's viscosity. We use a particularly simple relationship between temperature and viscosity: if the temperature is substantially below or above the melting point of the material, we leave the viscosity at a constant value. Within a temperature transition zone, we

[3]The variable viscosity formulation in this thesis, which is directly from [8], is in a simplified form where the viscous stress has been taken to be $\nabla \cdot (\nu \nabla \mathbf{u})$ with varying viscosity, ν. However, the full viscous stress for an incompressible fluid with variable viscosity is in fact $\nabla \cdot \tau = \nabla \cdot (\nu \nabla \mathbf{u} + \nu [\nabla \mathbf{u}]^T)$. The present simplification differs from the full stress where the spatial gradients of ν are large. The missing terms and their rheological implications are discussed briefly in section 4.5. I would like to thank Ron Fedkiw for telling me that our formulation had missing terms.

vary the viscosity as either a linear or quadratic function of temperature. Many materials, including wax, have a rapid transition from high to low viscosity when the material is heated to the melting point. Thus for our simulations of wax we make this transition zone quite narrow. This means that our materials remain rigid if they are cooler than the melting point, and then quickly liquefy at the appropriate temperature. So far as the solver goes, however, we could use almost any relationship between temperature and viscosity that we want. The key to simulating the proper behavior based on viscosity that changes spatially is to use the variable viscosity version of the diffusion term, and we now turn to this issue.

To understand the changes needed to allow variable viscosity, we return to the velocity diffusion equation. For expository purposes we will write these equations for a forward Euler integrator, and the appropriate changes to an implicit form are to be understood. Recall equation 42 for the momentum diffusion contribution in 3D with constant viscosity:

$$u_{i,j,k}^{new} = u_{i,j,k} + \frac{\nu \Delta t}{\Delta x^2}(u_{i-1,j,k} + u_{i,j-1,k} + u_{i,j,k-1} - 6u_{i,j,k} + u_{i+1,j,k} + u_{i,j+1,k} + u_{i,j,k+1}). \tag{56}$$

This equation assumes that viscosity ν is the same at all cells, so that this parameter may be placed outside the parenthesis. As depicted in figure 13, when viscosity varies across the fluid it should be considered a property of the cell centers or the cell edges that separate pairs of adjacent cell faces.

Let the viscosity variable at the center of the left cell face be written as $\nu_{i-1/2,j,k} = \nu(i\Delta x - \Delta x/2, j\Delta x, k\Delta x)$, using the same notation as equation 6. The correct finite difference formulation of equation 56, including variable viscosity ([68], section 19.2), becomes:

$$u_{i,j,k}^{new} = u_{i,j,k} + \frac{\Delta t}{\Delta x^2} \begin{pmatrix} \nu_{i-1,j,k}(u_{i-1,j,k} - u_{i,j,k}) + \\ \nu_{i,j,k}(u_{i+1,j,k} - u_{i,j,k}) + \\ \nu_{i-1/2,j-1/2,k}(u_{i,j-1,k} - u_{i,j,k}) + \\ \nu_{i-1/2,j+1/2,k}(u_{i,j+1,k} - u_{i,j,k}) + \\ \nu_{i-1/2,j,k-1/2}(u_{i,j,k-1} - u_{i,j,k}) + \\ \nu_{i-1/2,j,k+1/2}(u_{i,j,k+1} - u_{i,j,k}) \end{pmatrix} \tag{57}$$

There are similar equations for v and w. The resulting matrix equations stay symmetric, and after we make the appropriate changes to an implicit form we can use the same preconditioned conjugate gradient matrix solver as before.

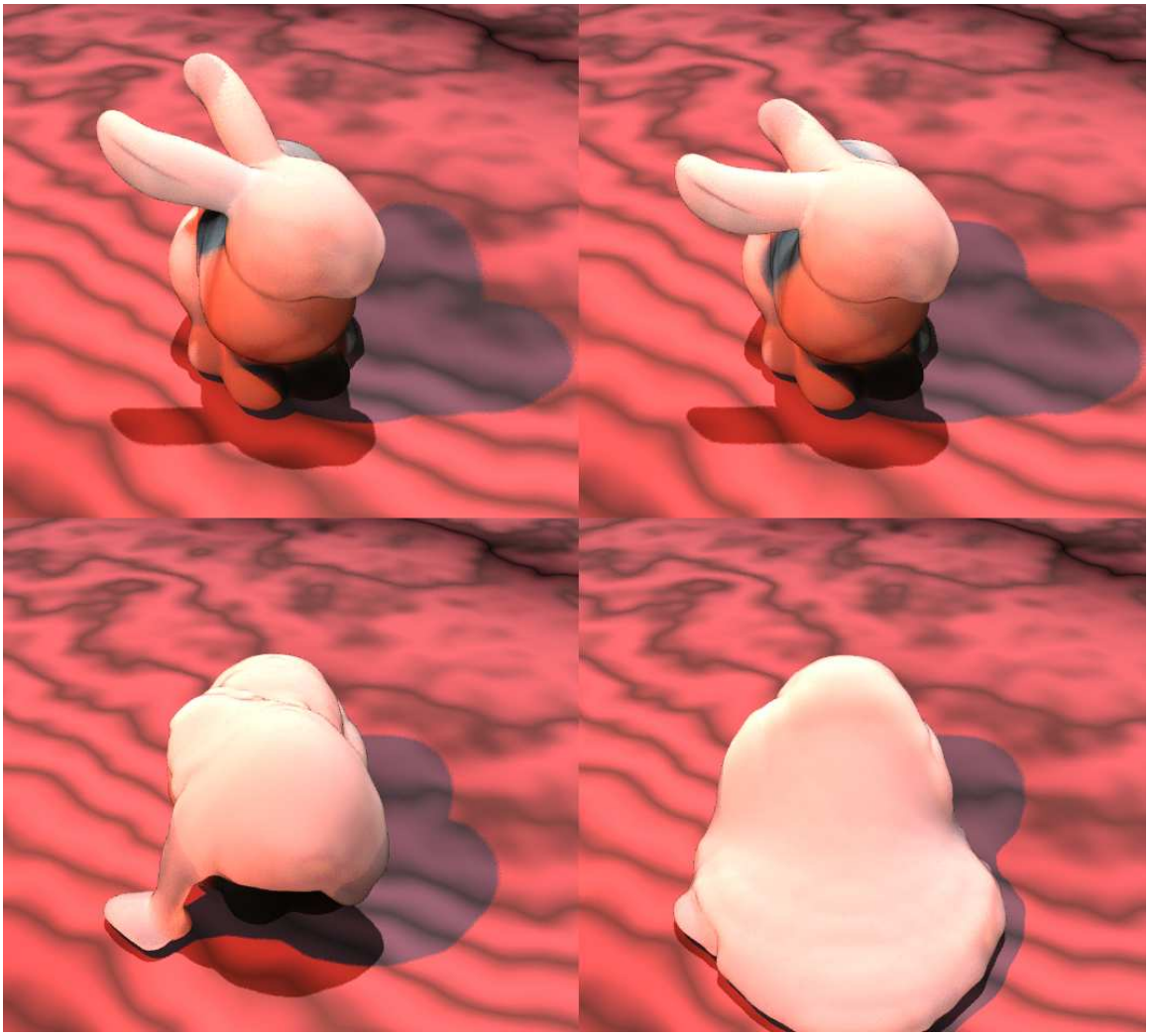

Figure 14: Melting bunny.

The above interpolated viscosities–$\nu_{i-1/2,j-1/2,k}$, $\nu_{i-1/2,j+1/2,k}$, $\nu_{i-1/2,j,k-1/2}$, and $\nu_{i-1/2,j,k+1/2}$– are at cell edge boundaries between cell faces (the red squared in Figure 13) and may be obtained by averaging the viscosities from the four cells that share the edge, since the material property controlling viscosity (*e.g.*, temperature) is identified with the cell centers, not the edges. However, when animating objects that melt, we get poor results when we use an arithmetic average between the four viscosities of the adjacent cells. The reason for this is that molten material that is dripping down the side of a rigid object will slow down more quickly because arithmetic viscosity averaging is dominated by the very large viscosity of the solid material. Rather than resort to higher grid resolutions, we found that a simple change alleviates this problem: if we use the geometric average (fourth root of the four-term product) instead of the arithmetic average when combining the viscosities of

adjacent cells, the lower viscosity dominates and the material continues to flow down the object's side. Thus we advocate using:

$$\nu_{i-1/2,j-1/2,k} = \left(\nu_{i,j,k}\nu_{i-1,j,k}\nu_{i-1,j-1,k}\nu_{i,j-1,k}\right)^{1/4} \tag{58}$$

to average the viscosity at location $(i\Delta x - \Delta x/2, j\Delta x - \Delta x/2, k\Delta x)$.

4.4 Results

We have used our fluid simulator to create several animations of viscous fluid and materials that melt or harden. Examples can be seen in the Figures. Figure 1 shows a block of wax that is being melted by a heat source near its upper right corner. Figures 11 and 14 show a similar wax-like simulation, but this time the model is the Stanford Bunny. This example demonstrates that our models may be given detailed geometry. Note that in a single time-step during this animation, one portion is entirely liquid (near the head) while an adjacent part is solid (the tail). Our solver gracefully handles such variations in viscosity.

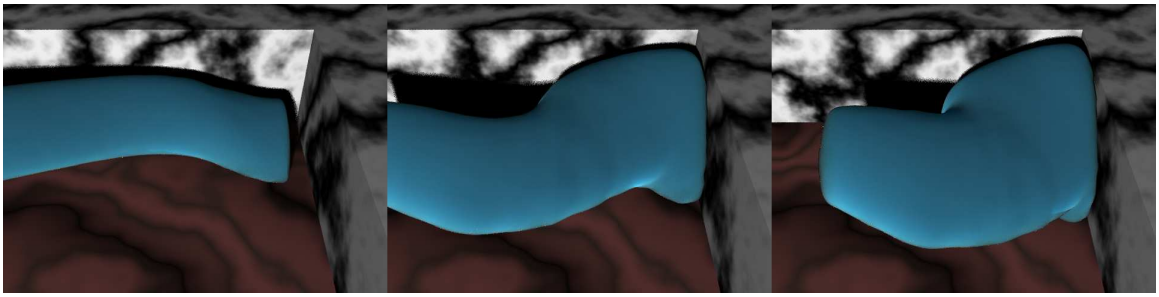

Figure 15: Toothpaste squirt.

The one figure I almost did not include in this thesis is figure 15, which shows a squirt of toothpaste hitting a wall. The interesting thing about this animation is how the toothpaste bends as it hits the wall.

The snapshots in Figure 12 demonstrate the behavior of fluid over a wide range of viscosity. Each column represents a different viscosity, from left to right: 0.1, 1, 10 and 100. Fluid has been thrown from above into a complex free-slip container that already holds a shallow pool of fluid. (The container walls are not rendered.) This example not only shows the difference between fluids with varying viscosity, but also demonstrates splashing and high velocity fluids.

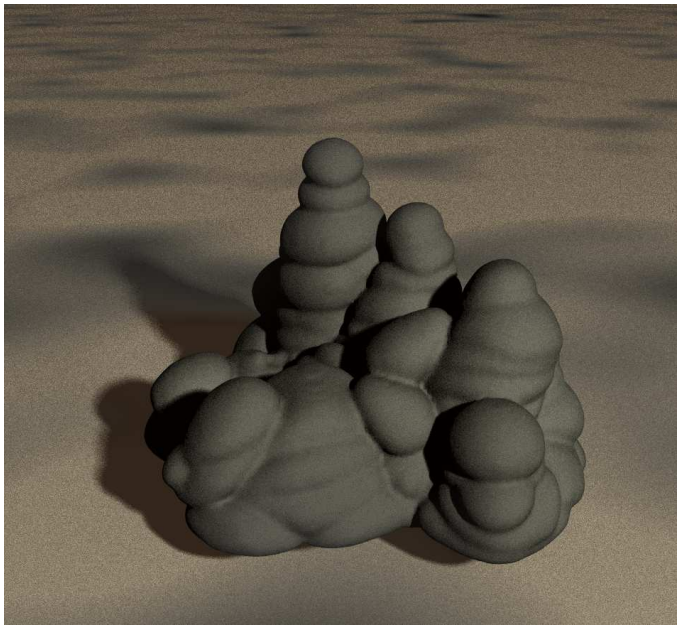

Figure 16: Drip sand castle.

Figure 16 shows an example of model creation. When very wet sand is dripped down onto the ground by a child at the beach, the drips of sand pile up to form sand castles. To simulate this, a user indicated locations and times for viscous spheres of sand to drop onto a ground plane. Because of the high viscosity, the simulated drops of sand do not melt together to form a large pool, but instead they pile up and retain their individual shapes.

4.5 Discussion

This chapter presents a technique for simulating materials that vary in viscosity from absolute rigidity to water-like. One possible way of doing so would be to have an arbitrary threshold between liquid and solid, and to treat these two cases individually. Instead, our approach is to model the range of material behaviors as variations in the viscosity of the material. We feel that this unified treatment of materials is the main contribution of [8] (the paper this chapter is heavily based on) to computer animation. The changes needed to implement this approach are straightforward to make to a MAC method fluid simulator (see chapter 3), and because of this we believe that others will have no trouble using our approach. The main additions to the MAC fluid solver presented in this chapter include high and variable viscosity simulated with a stable solver for liquids with free surfaces, and the the coupling of the liquids viscosity to a heat field that is diffused and advected with

the fluid. The approach allows us to rapidly animate liquids that are considerably more viscous than previously published graphics methods have allowed. By coupling the viscosity of a material to its temperature, we can animate objects that heat up, melt, flow, and harden. We would like to point out that in [8], there is a section on dampening of the velocity when fluid is in free flight. We have since come to believe that this was an error in the way we were setting our Dirichlet boundary conditions for the velocity in the viscosity solve.

The rest of this section is organized as follows. After commenting on the speed of our method, we discuss how our variable viscous diffusion differs from a true variable viscosity formulation for fluids. Finally, we discuss some possible future work.

4.5.1 Computation Time

The implicit integrator for velocity diffusion is stable even with large time steps, so our simulation times are fast. Table 6 shows simulation times for entire animation sequences. The melting bunny, for example, required about 0.55 seconds per frame of simulation time. This is dramatically faster than a forward Euler technique would allow.

Animation	**frame count**	**simulation time**	**viscosity**	**grid size**
Green Liquid 1	300	145	0.1	$32 \times 32 \times 32$
Green Liquid 2	300	104	1	$32 \times 32 \times 32$
Green Liquid 3	300	94	10	$32 \times 32 \times 32$
Green Liquid 4	300	94	100	$32 \times 32 \times 32$
Toothpaste	330	108	10,000	$42 \times 33 \times 18$
Bunny Melt	600	330	0.1 - 10,000	$35 \times 28 \times 38$
Drip Sand	750	397	50,000	$48 \times 48 \times 48$

Table 6: Simulation times (in seconds) are for entire animations, not for each frame. Simulations were run on a 2.0 GHz Pentium 4.

4.5.2 Viscous Stress Tensor

The variable viscosity formulation in section 4.3 is in a very specialized form. This section describes how the variable viscosity diffusion term was derived, what it is missing, and what affects the missing components may have on our fluid.

The viscous diffusion term, $\nu\nabla^2\mathbf{u}$, in equation 2, is a simplification of the viscous forcing term,

$\nabla \cdot \tau$, where τ is the viscous stress tensor,[4]

$$\tau = \begin{bmatrix} \nu(u_x+u_x) & \nu(v_x+u_y) & \nu(w_x+u_z) \\ \nu(v_x+u_y) & \nu(v_y+v_y) & \nu(w_y+v_z) \\ \nu(w_x+u_z) & \nu(w_y+v_z) & \nu(w_z+w_z) \end{bmatrix}, \tag{59}$$

and the viscous forcing term is the vector,

$$\begin{aligned} \nabla \cdot \tau &= \begin{bmatrix} \nu(u_{xx}+u_{yy}+u_{zz}+\{u_x+v_y+w_z\}_x)+\nu_x(u_x+u_x)+\nu_y(v_x+u_y)+\nu_z(w_x+u_z) \\ \nu(v_{xx}+v_{yy}+v_{zz}+\{u_x+v_y+w_z\}_y)+\nu_x(v_x+u_y)+\nu_y(v_y+v_y)+\nu_z(w_y+v_z) \\ \nu(w_{xx}+w_{yy}+w_{zz}+\{u_x+v_y+w_z\}_z)+\nu_x(w_x+u_z)+\nu_y(w_y+v_z)+\nu_z(w_z+w_z) \end{bmatrix} \\ &= \begin{bmatrix} \nu(u_{xx}+u_{yy}+u_{zz})+\nu_x(u_x+u_x)+\nu_y(v_x+u_y)+\nu_z(w_x+u_z) \\ \nu(v_{xx}+v_{yy}+v_{zz})+\nu_x(v_x+u_y)+\nu_y(v_y+v_y)+\nu_z(w_y+v_z) \\ \nu(w_{xx}+w_{yy}+w_{zz})+\nu_x(w_x+u_z)+\nu_y(w_y+v_z)+\nu_z(w_z+w_z) \end{bmatrix}, \end{aligned} \tag{60}$$

where the simplification is allowed because $\{u_x+v_y+w_z\}=0$ in an incompressible fluid. When the viscosity is constant in an incompressible fluid, the viscous forcing term, $\nabla \cdot \tau$, becomes the familiar viscous derivation term, $\nu\nabla^2\mathbf{u}$, because all of the partial derivatives of ν are zero.

When we derived our variable viscosity term we started with the already simplified viscous diffusion term instead of starting with the viscous forcing term. Our variable viscosity diffusion term was in fact $\nabla \cdot (\nu\nabla\mathbf{u})$ instead of $\nabla \cdot \tau$. This simplification is not noticeable in the portions of the fluid where variations in viscosity are small. However, in areas where the change in large, like when molten wax drips down the side of a candle, the missing terms are vital. The missing terms model the shearing viscous force inside the fluid. The use of a geometric mean (equation 58) does compensate somewhat for the large variations in viscosity, but it does not correctly model the viscous shearing force.

Rasmussen et al. [70] have a paper currently in press that uses a split explicit-implicit scheme to discretize the full viscous forcing term. However, their explicit terms are conditionally stable and force restrictions their time step that may be prohibitive.

[4]Because the fluid is incompressible, we ignore the bulk viscosity and the term, $-\mathbf{1}\frac{2}{3}\nabla \cdot \mathbf{u}$, in τ that is 0 when the fluid divergence is zero. For more information on the viscous stress tensor see [59] chapter 5.

4.5.3 Future Work

There are several topics that we are interested in pursuing in the future. One near-term topic is the texturing of models as they deform and flow. The sand texture of Figures 16 does not move with the surface, and we seek a method of making the texture "stick" to the model. Another extension would be to use the level-set method of [24] instead of particle splatting to define the surface of the fluid. Chapter 6 of this thesis explores some of the observations and difficulties we came across while making the transition to a level set representation for out free surface. To allow large time-steps for fast moving fluid, it would also be necessary to use another method for the convection term such as the semi-Lagrangian approach given in [79, 24]. The rigid fluid method in Chapter 5 uses the semi-Lagrangian technique and we found that it is a very simple technique to use, though it does incur additional numerical dampening to the advection process. A more long-term research question is how to allow the cracking of material that has melted and then hardened, perhaps through the inclusion of surface tension or viscoelastic forces [32]. Many materials such as mud and lava crack while they harden, and it would be wonderful to animate this process.

CHAPTER V

RIGID FLUID

"*NON-ULTRABUOYANT*? I THINK I MADE UP A WORD.
NOT THE BEST SCIENCE WORD I'VE EVER MADE
UP, THAT HONOR IS RESERVED FOR *DIFFUSIVITIVELY*"

Peter J. Mucha

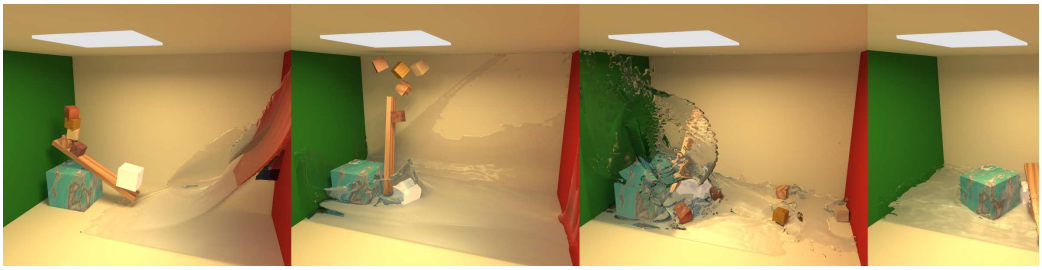

Figure 17: A silver block catapulting some wooden blocks into an oncoming wall of water.

In the last chapter solid objects were treated as fluid with very high viscosity. This chapter also treats solids as if they were fluid, but instead of changing the viscosity of the fluid, the *velocities* of the fluid inside the solid object are changed. The technique in this chapter, know as the *Rigid Fluid* method, gets its name from the way those velocities inside the solid are changed (*i.e.*, constrained) to be rigid body velocities. The rigid fluid method most closely follows the *distributed Lagrange multiplier* (DLM) technique as described by Patankar [60]. The DLM technique is so named because the constraints that enforce rigidity are distributed over the solid domain. However, there are distinct differences that make the rigid fluid method unique. The rigid fluid method uses the finite difference MAC method with pressure projection that is popular in the current computer graphics literature, instead of the more complicated finite element method that Patankar uses. Also, the rigid bodies in the rigid fluid method can be any polygonal object and are not restricted to spheres, as they are in [60]; because of this the rigid fluid formulation also allows for torques because it can apply any force at any point on the rigid body, not just repulsion forces at the center of mass. The

rigid fluid technique also incorporates free surfaces via level sets [14].

5.1 Rigid Fluid Domains

The Navier-Stokes equations which govern the movement of an incompressible fluid are covered in chapter 3, and the extension to the implicit viscous Navier-Stokes equations are described in chapter 4. Before discussing the equations that govern the rigid fluid method this section introduces some notation for the computational domain and the next section describes the *deformation operator*, $\mathbf{D}[\ ]$.

Figure 18: The left side of this figure is the computational domain, and the right is the rendered frame. On the left the yellow area is the fluid domain $\mathbb{F}$; the blue is the rigid body domain $\mathbb{R}$. Notice that the small blocks on the right are not touching liquid, so they will be controlled by the rigid body solver until they touch liquid.

There are two parts to the computational domain, depicted in figure 18. The part of the domain containing only the fluid is $\mathbb{F}$, and the union of cells occupied by the rigid bodies is the solid domain, $\mathbb{R}$. The two domains are disjoint, and together they form the complete computational domain, $\mathbb{C} = \mathbb{F} \cup \mathbb{R}$. The boundary separating $\mathbb{F}$ and $\mathbb{R}$ is $\partial\mathbb{R}$.

5.2 Rigid Body Motion as Constraints

As mentioned above there are two parts to the computational domain, depicted in figure 18. This section describes what makes the solid domain, $\mathbb{R}$ different from the fluid domain. The Navier-Stokes equations are solved in both domains, but there is an extra *rigidity* constraint enforced on $\mathbb{R}$

with the *deformation operator*, $\mathbf{D}[\ ]$. This section describes the properties of rigid body motion, and how $\mathbf{D}[\ ]$ can be used to express that motion. The use of $\mathbf{D}[\ ]$ to quickly enforce rigid body motion in the solid domain was first proposed by Patankar et al. [61].

When simulating a rigid body, it is useful to think of its motion as translations and rotations about the rigid body's center of mass. Simplifying the motion of all points in the solid object with these assumptions hides the complexity of its rigid body motion. In essence, rigid body solvers implicity enforce the *rigidity* of the solid by constraining its motion to translations and rotations about the center of mass.

The rigid fluid method, on the other hand, solves the equations of motion for the rigid bodies with the Navier-Stokes equations (equations 1 and 2), the Navier-Stokes equations allow all sorts of deformations in the velocity of the fluid, so the *rigidity* of the rigid bodies must be explicitly enforced with a Lagrange multiplier. The rigidity constraint enforced in the rigid body domain is very much like the incompressibility constraint discussed in section 3.2.4. The rigidity constraint, however, is a stricter constraint, as it is both divergence free, like the incompressibility constraint, and *deformation free*. The rigidity constraint dictates that for every point $\mathbf{y}_j$ in a rigid body, the following relationship must hold:

$$\dot{\mathbf{y}}_j = \mathbf{v} \ + \ \omega \times \mathbf{r}_j \tag{61}$$

for some constant $\mathbf{v}$ and ω. In the above equation, $\dot{\mathbf{y}}_j$ is the velocity at $\mathbf{y}_j$, $\mathbf{r}_j$ is a vector pointing from the rigid body's center of mass, $\mathbf{x}$, to $\mathbf{y}_j$, $\mathbf{v}$ is the translational velocity at $\mathbf{x}$, and ω is the angular velocity about $\mathbf{x}$ along the axis $\omega/|\omega|$ with magnitude $|\omega|$.

The rigidity constraint can be expressed by means of the deformation operator, $\mathbf{D}[\ ]$, defined for any vector field $\mathbf{u} = (u,v,w)$ by

$$\begin{aligned} \mathbf{D}[\mathbf{u}] &= \frac{1}{2}[\nabla\mathbf{u} + \nabla\mathbf{u}^T] \\ &= \frac{1}{2}\begin{bmatrix} 2u_x & (v_x+u_y) & (w_x+u_z) \\ (v_x+u_y) & 2v_y & (w_y+v_z) \\ (w_x+u_z) & (w_y+v_z) & 2w_z \end{bmatrix}. \end{aligned} \tag{62}$$

The 3×3 symmetric tensor $\mathbf{D}[\mathbf{u}]$ measures the spatial deformation of $\mathbf{u}$. The constraint,

$$\mathbf{D}[\mathbf{u}] = \mathbf{0} \quad \text{in } \mathbb{R}, \tag{63}$$

ensures the motion in $\mathbb{R}$ is in fact rigid body motion [61]. It does not tell us what that motion is, but we know it must agree with equation 61, that is, if we know the position of the points where $\mathbf{u}$ is defined and their relative positions to the rigid body's center of mass, then there is some known translational velocity, $\mathbf{v}$, and some rotational velocity, ω, that we can find for that rigid body. Depending on the solution procedure employed, it can be advantageous [31] to use an equivalent differential relationship in place of equation 63.[1]

5.3 Rigid Fluid Governing Equation

In this section we present the governing equations that form the heart of the rigid fluid method. Similar equations were originally derived in [31] for the DLM method in the weak form appropriate for finite element computations. In contrast, we use a strong form appropriate for finite differences. To streamline our exposition we assume in this section that there is no viscoelastic stress, and all boundary conditions except those on the interface between the rigid bodies and the fluid are assumed to be understood. If viscoelastic stress is needed, then the ${\rho_f}^{-1}\nabla\, p$ term in equation 64 should be changed to ${\rho_f}^{-1}\nabla\cdot(\Sigma - p\,\mathbf{1})$ and Σ should be added to the second part of equation 67. Σ is the extra stress tensor that depends on the deformation rate and deformation history of the fluid at a specific location.

Recalling the definitions of $\mathbb{C}$ and $\mathbf{D}[\,]$ from the previous sections we are ready to describe the governing equations for the rigid fluid method. The conservation of momentum equations are defined as

$$\mathbf{u}_t = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} - \frac{1}{\rho_f}\nabla p + \mathbf{f} \quad \text{in } \mathbb{F} \tag{64}$$

in the fluid domain (same as equation 2), and

$$\mathbf{u}_t = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla\cdot\Pi - \frac{1}{\rho_r}\nabla p + \mathbf{f} \quad \text{in } \mathbb{R} \tag{65}$$

in the solid domain, where ρ_f is the mass density of the fluid, and ρ_r is the density of the rigid body. The viscous diffusion term is absent in equation 65 because the rigidity constraint already eliminates Newtonian viscous dissipation, however there is an extra term due to the deformation

[1] The differential relationship would be $\nabla\cdot(\mathbf{D}[\mathbf{u}]) = 0$ in $\mathbb{R}$ and $\mathbf{D}[\mathbf{u}]\cdot\mathbf{n} = 0$ on $\partial\mathbb{R}$.

stress inside the solid that is required to maintain rigidity. We implicitly define Π as that extra part of the deformation stress in addition to the harmonic pressure field, p.

Since the deformation-free constraint we enforce on the rigid body domain with equation 63 is stronger than the divergence-free one, we can, for convenience, enforce the divergence-free constraint over the entire domain with

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \mathbb{C}. \tag{66}$$

The constraints realized by equations 66 and 63 will be enforced by projections onto divergence-free and deformation-free motion in the appropriate domains.

The no-slip and dynamic force boundary conditions between the solids and the fluid are defined as

$$\mathbf{u} = \mathbf{u}_i \quad \text{and} \quad (2\rho_f \nu \mathbf{D}[\mathbf{u}] - p\mathbf{1}) \cdot \mathbf{n} = \mathbf{t} \quad \text{on } \partial\mathbb{R} \tag{67}$$

where $\mathbf{1}$ is the identity tensor, $\mathbf{u}_i$ and $\mathbf{n}$ are the velocity and normal on $\partial\mathbb{R}$, and $\mathbf{t}$ is the traction force of the fluid on the solid as a sum of the projected viscous stress and pressure. A similar condition can be written for the force of the solid on the fluid in terms of the solid stresses, which must be equal and opposite to $\mathbf{t}$ by Newton's third law. However, we will never need to directly enforce the boundary conditions in equation 67, as they will be approximately captured by the projection techniques described below.

5.4 Rigid Fluid Implementation

Equations 63 through 67 are the governing equations for all the moving objects in our simulation, both solid and fluid. We solve these equations in three steps. First, we solve the Navier-Stokes equations 1 and 2 for the entire domain $\mathbb{C} = \mathbb{F} \cup \mathbb{R}$. During this first step, the rigid objects are treated *exactly as if they were fluid*. Next, we calculate the rigid body forces due to collisions and relative density add those forces to the grid locations inside the rigid body domain. Finally, we enforce rigid motion for the velocities at those grid locations inside each solid object. These three steps move the simulation forward in time, from $\mathbf{u}^n \rightarrow \mathbf{u}^{n+1}$, passing through two successive intermediate stages $\mathbf{u}^*$ and $\hat{\mathbf{u}}$ along the way. In this section we will ignore issues of immobile walls and moving the fluid/air interface, which we will return to in section 5.5.

5.4.1 Solving Navier-Stokes Equations: $\mathbf{u}^n \rightarrow \mathbf{u}^*$

We first solve the Navier-Stokes equations using an operator splitting scheme (see section 4.1.1) over all initial velocities $\mathbf{u}^n$ in $\mathbb{C}$ with four steps:

1. We add the body force, $\Delta t\mathbf{f}$, to all of $\mathbb{C}$. Because rigid body motion will be enforced only inside $\mathbb{R}$, there will be a slip error at $\partial\mathbb{R}$ that increases as $|\rho_r - \rho_f|$ increases. One way to reduce this error, suggested by Patankar [60], is to add the extra buoyant-weight term, $\Delta t\mathbf{f}(\rho_r - \rho_f)/\rho_f$, to $\mathbb{R}$ at this step. If this is done then $\mathbf{f}$ must be removed from equation 70.

2. We solve the advection term, $-(\mathbf{u}\cdot\nabla)\mathbf{u}$, using the semi-Lagrangian technique which Stam introduced to the graphics community [79].

3. We solve the diffusion term, $\nu\nabla^2\mathbf{u}$, using the implicit viscosity formulation in [8]; however, we corrected the Dirichlet boundary conditions at the free surface so that we do not cause the velocity dissipation discussed in that paper.

4. We use pressure projection (section 3.2.4) to make the velocity in $\mathbb{C}$ divergence free. Because the semi-Lagrangian technique behaves better on a divergence-free velocity field, we have the option in our code to use pressure projection before the advection step in addition to here. All the animations in this chapter use this option. Other options are to solve the advection term first and then add the body forces and solve the viscous terms, or use a conditionally stable yet more accurate solver for advection.

Each of the above steps is stable for large time steps, even with stiff viscous effects, and the only criteria we have to adhere to is the CFL condition. Upon completion of these steps we have the divergence free velocity field $\mathbf{u}^*$ in $\mathbb{C}$, but it is not the final velocity field because we have not accounted for collision and relative density forces of the rigid bodies, nor has the velocity in $\mathbb{R}$ been constrained to rigid body motion.

5.4.2 Calculating Rigid Body Forces: $\mathbf{u}^* \rightarrow \hat{\mathbf{u}}$

During the time step, the rigid body solver applies collision forces to the solid objects as it updates their positions. These forces must be included in the velocity field to properly transfer momentum between the solid and fluid domains.

As each collision force, $\mathbf{F}_j$, is applied to one of the $\mathtt{N}$ rigid bodies, we keep a running sum of the accelerations created on that body over the time step and store it as

$$\mathbf{A}_c = \sum_j \frac{\mathbf{F}_j}{M_i}, \tag{68}$$

where $i \in \{1, 2, \ldots, \mathtt{N}\}$, and M_i is the mass of the rigid body to which the force is applied.

Similarly, as each force is applied at point $\mathbf{p}_j$, we sum the angular accelerations it creates about each body's center of mass,

$$\alpha_c = \sum_j \mathbf{I}_i^{-1}[\ (\mathbf{p}_j - \mathbf{x}_i) \times \mathbf{F}_j\ ], \tag{69}$$

where $\mathbf{I}_i$ is the moment or inertia of the i^{th} rigid body, in its current orientation, and $\mathbf{x}_i$ is its center of mass.

Forces that arise from the *relative density*, also known as the specific gravity, must also be considered. The relative density of a solid is the ratio of its density to that of the surrounding fluid, ρ_r/ρ_f. If the relative density is greater than 1, then the solid will sink. Conversely, if the relative density is less than 1 the solid will rise and float. It becomes more difficult for the fluid to move an object as the relative density increases. The relative density and collision forces are accounted for in $\mathbb{R}$ with a momentum source term

$$\mathbf{S} = \rho_r \mathbf{A}_c + \mathbf{r}_i \times \rho_r \alpha_c - (\rho_r - \rho_f)[\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + (\mathbf{u}^* \cdot \nabla)\mathbf{u}^* - \mathbf{f}\ ], \tag{70}$$

where the vectors $\mathbf{r}_i = \mathbf{y}_i - \mathbf{x}_i$ point from the center of mass of the i^{th} rigid body to the grid point locations, $\mathbf{y}_i$, in that rigid bodies domain, $\mathbb{R}_i$. The solution to equation 70 is direct because all the variables on the right hand side are known.

The relative density term in equation 70 is due to Patankar [60]. Since he restricts his attention to spherical objects with repulsion forces acting only at the center of mass, Patankar includes an $\mathbf{A}_c$, but not an α_c, collision term. The angular collision terms α_c are essential for the proper treatment of non-spherical objects with collision forces that generate torques.

Using $\mathbf{S}$ we solve for a new velocity field,

$$\hat{\mathbf{u}} = \mathbf{u}^* + \ w\frac{\Delta t}{\rho_r}\mathbf{S}, \tag{71}$$

where w is a number between 0 and 1 representing the fraction of volume of a computational cell occupied by the solid.

The momentum conserving velocity, $\hat{\mathbf{u}}$, is still not rigid body motion, so we must complete one last step.

5.4.3 Enforcing Rigid Motion: $\hat{\mathbf{u}} \rightarrow \mathbf{u}^{n+1}$

To guarantee rigid body motion in $\mathbb{R}$, the unknown force, $\mathbf{R}$, that maintains rigidity must be found. Once found, the final velocity,

$$\mathbf{u}^{n+1} = \hat{\mathbf{u}} + \frac{\Delta t}{\rho_r}\mathbf{R}, \tag{72}$$

can be solved for in $\mathbb{R}$, but we still do not have an equation for $\mathbf{R}$.

Equation 72 is a projection that enforces the rigidity constraint in much the same way the pressure projection, equation 29, enforces the divergence-free constraint. The constraint enforced by the pressure projection was $\nabla \cdot \mathbf{u} = 0$, and the Lagrange multiplier used to enforce that constraint was p. So, to find an equation for p we took the divergence of equation 29 and arrived at equation 31. The constraint that must be enforced with the rigidity projection is equation 63, and the Lagrange multiplier is $\mathbf{R}$, so substituting equation 72 into equation 63 yields an equation for $\mathbf{R}$:

$$\mathbf{D}[\mathbf{u}^{n+1}] = \mathbf{D}[\hat{\mathbf{u}} + \frac{\Delta t}{\rho_r}\mathbf{R}] = 0 \tag{73}$$

which states $\hat{\mathbf{u}} + \Delta t \mathbf{R}/\rho_r$ is the desired rigid body motion. Alternatively, we break $\hat{\mathbf{u}}$ in $\mathbb{R}$ into two parts:

$$\hat{\mathbf{u}} = \hat{\mathbf{u}}_R + \hat{\mathbf{u}}', \tag{74}$$

where $\hat{\mathbf{u}}_R$ is the rigid body velocity we are searching for, and

$$\hat{\mathbf{u}}' = -\frac{\Delta t}{\rho_r}\mathbf{R} \tag{75}$$

is due to the stress inside $\mathbb{R}$ that enforces rigid body motion upon it.

As observed by Patankar [60], the desired rigid body solution of equation 73 and 75 for $\mathbf{R}$ and $\hat{\mathbf{u}}'$ must conserve momentum and can therefore be obtained directly. Writing equation 61 as a union over each rigid body yields the equation

$$\hat{\mathbf{u}}_R = \bigcup_i (\hat{\mathbf{v}}_i + \hat{\omega}_i \times \mathbf{r}_i) \tag{76}$$

for some $\hat{\mathbf{v}}_i$ and $\hat{\omega}_i$. Because momentum must be conserved, we obtain $\hat{\mathbf{v}}_i$ and $\hat{\omega}_i$ for each rigid body by directly integrating the intermediate $\hat{\mathbf{u}}$ inside a given rigid body $\mathbb{R}_i$ with the equations:

$$M_i\hat{\mathbf{v}}_i = \int_{\mathbb{R}_i} \rho_i\hat{\mathbf{u}}\,dy_i, \qquad \text{and} \tag{77}$$

$$\mathbf{I}_i\hat{\omega}_i = \int_{\mathbb{R}_i} \mathbf{r}_i \times \rho_i\hat{\mathbf{u}}\,dy_i, \tag{78}$$

where M_i, $\mathbf{I}_i$ and ρ_i are the mass, moment of inertia and density of the i^{th} rigid body, and dy_i is the volume of the grid cell occupied by the solid:

$$dy = \Delta x^3 w. \tag{79}$$

Equations 77 and 78 are evaluated by summing the appropriate terms for each grid cell that is fully or partially inside the i^{th} rigid body domain $\mathbb{R}_i$.

Because equation 72 must be a momentum conserving projection, we simply use equation 77 and 78 directly to solve for the rigid body velocity $\hat{\mathbf{u}}_R$. We then distribute this rigid body velocity over the objects to get our final velocity:

$$\mathbf{u}^{n+1} = (1-w)\hat{\mathbf{u}} + w\hat{\mathbf{u}}_R, \tag{80}$$

which enforces rigidity and conserves momentum inside $\mathbb{R}$.

5.5 Advancing the Computational Domain

The rigid fluid method uses a regularly spaced discrete computational domain where the components of the velocity vector are on the faces of the grid cells, and the pressure is in the center; this is the MAC grid domain as described in section 3.2.1. The fluid and solid domains are advanced each time step, so before we can solve the rigid fluid equations we must determine the new computational domain and identify the grid cells in $\mathbb{F}$ as well as the grid cells in $\mathbb{R}$. We must also identify the space that $\mathbb{F}$ and $\mathbb{R}$ can not occupy–the immobile boundaries. A static signed distance function, similar to the one used in [103], delineates the immobile boundary from the rest of the computational domain. The immobile boundary region has a velocity, so it can be used for objects with one-way solid-to-fluid coupling if the user desires. None of the animations in this work have that type of one-way solid-to-fluid coupling, but interested readers can find out more about it in [24].

To advance the computational domain, $\mathbb{C}$, we must advance both $\mathbb{F}$ and $\mathbb{R}$. We will describe the advancement of the fluid domain, followed by the advancement of the solid domain.

We use the particle level set, ϕ, to identify the fluid region [14]. In the level set implementation, all grid cells have a width of 1 and the time step is assumed to be 1 for simplicity, but the grid cells in $\mathbb{C}$ have a width of Δx and will be advanced by Δt. Also, the level set exists on a regular grid, while the velocity in $\mathbb{C}$ is solved on a *staggered grid* (section 3.2.1). Therefore, we compute the velocity for the level set by averaging the velocities at the faces of the cell, equation 6, and then scaling that average by $\Delta t/\Delta x$. This not only simplifies the level set implementation, it also decouples the time step restrictions of the level set equation from the rest of the simulation. An extension velocity is grown into the empty air regions not filled by the averaging to realistically advance the level set. See chapter 6 for details on the extension velocities and level set implementation.

Once the level set position is updated, the rigid fluid method identifies the new $\mathbb{R}$ by moving the rigid bodies with the solver described in [36]. The rigid bodies are polygonal objects, possibly concave, and their mass properties are computed directly from the polygonal representation [51]. Before the rigid body solver can take a step it must decide which solids are touched by the fluid and which are not. Those that are not touching fluid are updated exactly as in [36], but those that are touching fluid must obtain their velocity by integrating with equations 77 and 78.[2] Let us examine these equations in more detail for a single rigid body in domain $\mathbb{R}$, with a mass M, a density ρ, a moment of inertia $\mathbf{I}$, and a center of mass $\mathbf{x}$:

$$M\mathbf{v} = \int_{\mathbb{R}} \rho \mathbf{u}\, dy, \qquad \text{and} \tag{81}$$

$$\mathbf{I}\omega = \int_{\mathbb{R}} \mathbf{r} \times \rho \mathbf{u}\, dy, \tag{82}$$

where $\mathbf{r}$=$\mathbf{y}-\mathbf{x}$, and $\mathbf{y}$ is the current point in $\mathbb{R}$ that is under integration. The purpose of these two equations is to find the current translational velocity $\mathbf{v}$ and angular velocity ω, given the fluid velocity $\mathbf{u}$. Several steps are involved in calculating the discrete integrals from the above equations 81 and 82. First a volume and center of mass is stored for each piece of the solid that is contained

[2] In [36] the updated rigid body velocity, $\mathbf{v} = \mathbf{v} + \Delta t\mathbf{f}$, is used to check for interference. When the rigid body is wet we simply use the old velocity instead of an updated one. The most accurate way to update wet rigid bodies would be to calculate the updated velocity with equations 77 and 78, but this would require a lot of re-calculation and back-tracking of fluid equations, and the fluid equation is far too expensive to solve so liberally.

in a cell. Second, the discrete moment of inertial and mass are calculated, and while gathering the data for the second part, the equations are solved. Let us look at each of these steps in more detail.

In section 5.1 we mentioned that $\mathbb{R}$ is a collection of cells that contain part of a solid. In fact, once a solid has been moved to a new position, we keep a list of the cells that it occupies along with some important information. Two important things to know per cell are the volume of the rigid body that occupies that cell, and the center of mass of that small piece of the rigid body. In an early implementation of the rigid fluid method, we actually took the intersection of the rigid body triangles and the cell, then used that new polygonal object to calculate an exact center of mass and volume for it [51]. We note that this also gave us an accurate moment of inertia for that piece as well. As the rigid bodies we wished to simulate became more complicated, we realized that this approach was over-kill, and error prone. However, with good data structures, and skillful programmers who are willing to take time designing them, exact polygonal representations of the solid pieces in the cell will give a more accurate discrete integral. We decided to use a simpler approach using signed distances, however, and found that it was adequate.

Each rigid body is given a signed distance representation. One drawback to the signed distance representation is that it must be held in memory, so if many complicated rigid bodies are needed in the scene we suggest using a sparse data structure to store them [29]. The advantage of this representation is constant time inside/outside tests. Negative values are inside the object and positive values are outside the object. Each center of the cell is tested, and if the distance is greater than the distance to the corner of the cell from the center (*i.e.*, $> \sqrt{3}\Delta x/2$) then there is zero volume in the cell. If the distance is less than $-\sqrt{3}\Delta x/2$, then the cell is entirely inside the rigid body, the volume of the cell is Δx^3, and the center of mass is the center of the cell. We have found that for simple objects like sphere and cubes, approximating the volume with the level set value at the cell center works well (*e.g.*, if $\phi = 0$ then the volume is $0.5\Delta x^3$). The center of mass, in this case, can still be the center of the cell. This simple treatment, however, works poorly when the MAC grid is coarse compared to the object. In that case, like with the bunny model which has ears that are only a couple grid cells wide, we subdivide the grid twice into 64 smaller cells. We test the center of these cells to get a more accurate volume and center of mass.

With the volume and center of mass calculated for each cell that the solid intersects, the discrete

```
M = 0.0
I = 0.0
Mv = 0.0
Iomega = 0.0
for(n = 0; n < N; n++)
{
    m[n] = volume[n]*ρ
    r[n] = x[n] - x
    M += m[n]
    Mv += m[n]*u[n]
    I[n] = [ r[n].y*r[n].y + r[n].z*r[n].z        -r[n].x*r[n].y                -r[n].x*r[n].z
             -r[n].x*r[n].y          r[n].x*r[n].x + r[n].z*r[n].z        -r[n].y*r[n].z
             -r[n].x*r[n].z                -r[n].y*r[n].z          r[n].x*r[n].x + r[n].y*r[n].y ]
    I += I[n]
    Iomega += m[n]*(r[n]×u[n])
}
v = Mv/M
ω = Iomega/I /* matrix inversion with LU decomposition [33], p.88,121 */
```

Table 7: Pseudo-code to calculate the discrete integrals in equations 81 and 82.

mass is simply the sum of all the volumes multiplied by the objects density. If variable density objects are desired then the density field must also be stored. The moment of inertia is more complicated. Each of the $n \in \{1,2,\ldots,\mathtt{N}\}$ cells occupied by the solid has a mass, m_n and a center of mass, $\mathbf{x}_n$. Using the rigid body's known center of mass, $\mathbf{x}$, we calculate the position vector $\mathbf{r}_n = \mathbf{x}_n - \mathbf{x} = (x_n, y_n, z_n)$, which points from the true center of mass of the rigid body to the center of mass of the piece of the rigid body occupied by cell n. The discrete moment of inertia for a single cell, $\mathbf{I}_n$, is then:

$$\mathbf{I}_n = m_n \begin{bmatrix} y_n^2 + z_n^2 & -x_n y_n & -x_n z_n \\ -x_n y_n & x_n^2 + z_n^2 & -y_n z_n \\ -x_n z_n & -y_n z_n & x_n^2 + y_n^2 \end{bmatrix}. \tag{83}$$

Once the velocities are calculated via the last two lines of the algorithm in table 7, the velocities from equations 77 and 78 are used as initial conditions to the rigid body solver. During one rigid body time step, forces and torques are applied to the rigid bodies if there are collisions. As the collision forces are applied, a running sum of the accelerations and angular accelerations they create are kept (equations 68 and 69) and stored in $\mathbf{A}_c$ and α_c. The variables $\mathbf{A}_c$ and α_c represent the accelerations of the rigid bodies due to collisions and are used in equation 70. Strictly speaking, the force, $\mathbf{F}$, used in equations 68 and 69 has dimensions $mass * space/time^2$, but the impulse-based

rigid body solver we use [36] gives us *impulse forces* with dimensions $mass * space/time$. We change the impulse forces to standard forces by dividing them by the current time step Δt. This chapter, and our code use standard forces because we did not wish to restrict the rigid body method to impulse based rigid body solvers.

As mentioned above, after a new position is found for each of the rigid bodies, we save a list of the grid points that are inside each solid along with a per-cell volume and center of mass. The reason we store a per-cell volume and not a per-cell mass is because the density of the rigid body may vary with space or time, and keeping the stored per-cell quantities as general as possible helps simplify changes we may wish to make at a later date. Just as in [36], we have a signed distance function for each of our rigid bodies. Simple shapes like spheres and boxes use an exact formula, very complex shapes used a static level set, and shapes with few triangles, like the gems in Figure 21, use a parity ray-shooting test. However it is implemented, this signed distance function is important because it affords fast (constant time except for the parity test) inside/outside tests of the rigid bodies. The speed of this test is important because large objects can take up many grid cells, and for each grid node in a rigid body several inside/outside tests may be necessary to calculate the discrete moment of inertia and mass described above.

Once we find $\mathbb{R}$ and the list of grid points that reside within it, the fluid domain, $\mathbb{F}$, is found as any grid point not in $\mathbb{R}$ and with $\phi \leq 1/2$.

5.6 Results

In this section, we describe some animations that were created using the rigid fluid technique. Figure 17 shows an animation where a silver block, measuring 20cm in each dimension and with a relative density of 9, is dropped from the top of a one meter tall room. It strikes a plank of wood (relative density 0.74) and catapults several smaller wooden blocks into an oncoming wall of water. The silver block is heavy enough not to move much so the water is forced over and around it. The turquoise block has eight times the volume of the silver block, but its relative density is only two, so it slides around more. Some things to notice in this animation include how the light wooden blocks bob up and down on the surface of the water. The small blocks do cause splashes when they land in the water, but the water also pushes them around so they move with the swells of the sloshing water.

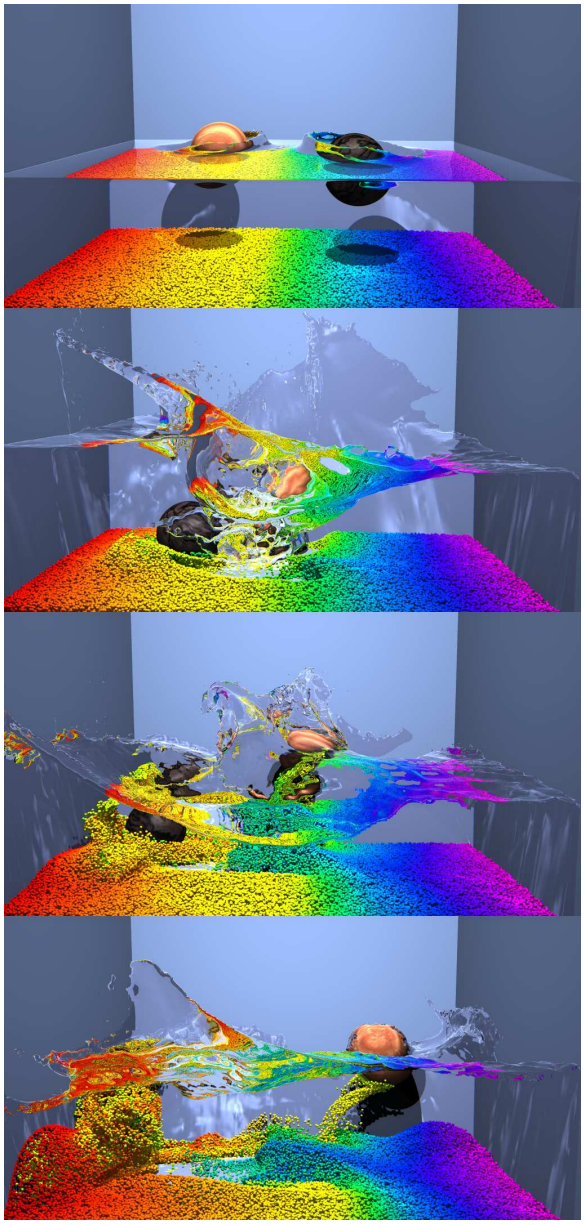

Figure 19: A lead and wood ball from figure 2 with passively advected particles.

In figures 2 and 19 a lead and a wood sphere (densities 11 and 0.55) are thrown into a meter wide pool. The spheres have the same initial downward velocity of 3.1m/s, equal but opposite horizontal velocities of 3.8m/s, and are rotating in opposite directions at 1000rpms about their vertical axis. As revealed by their shadows, the spheres are slightly off center from one another at the start of the simulation but will obviously strike one another. The lead sphere is heavy enough not to be moved much by the wood sphere, and its angular momentum rolls it into the back left corner of the tank as expected, but the lead sphere strikes the wood one hard enough to drive it against the wall.

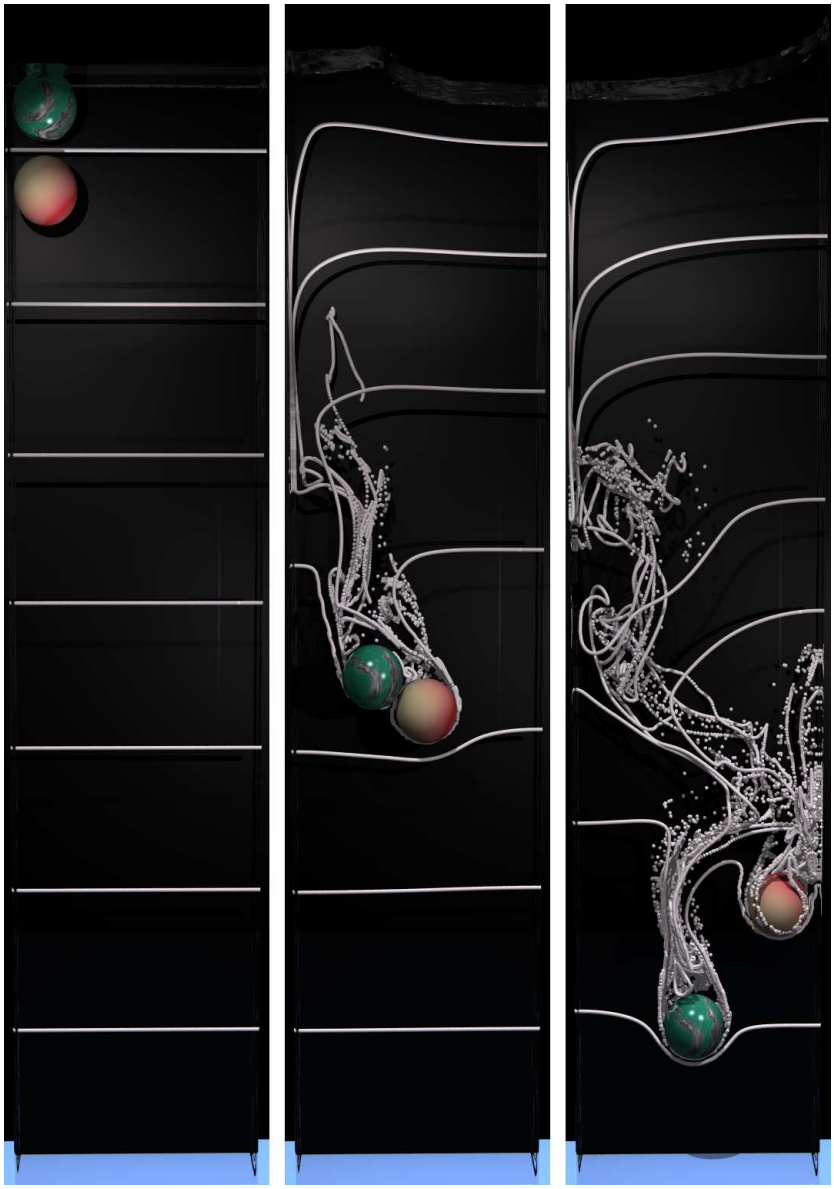

Figure 20: Tumbling of two sinking stones. Passively advected particles mark the fluid movement.

Physical experiments have shown that when two spheres sink in a tank of liquid, one placed just above the other, a phenomenon occurs known as “drafting, kissing, and tumbling.” We tested this with the rigid fluid technique and were easily able to reproduce the effect. As shown in figure 20, we placed two spherical stones (radius 8.25cm and relative density 5.7) at the top left of a three meter tall tank of water. Notice that the turquoise stone starts above the marble stone. As they sink together, they drift closer, and eventually tumble around one another. Because of the tumbling, the turquoise stone reaches the bottom first, even though it started on top. We observe qualitatively

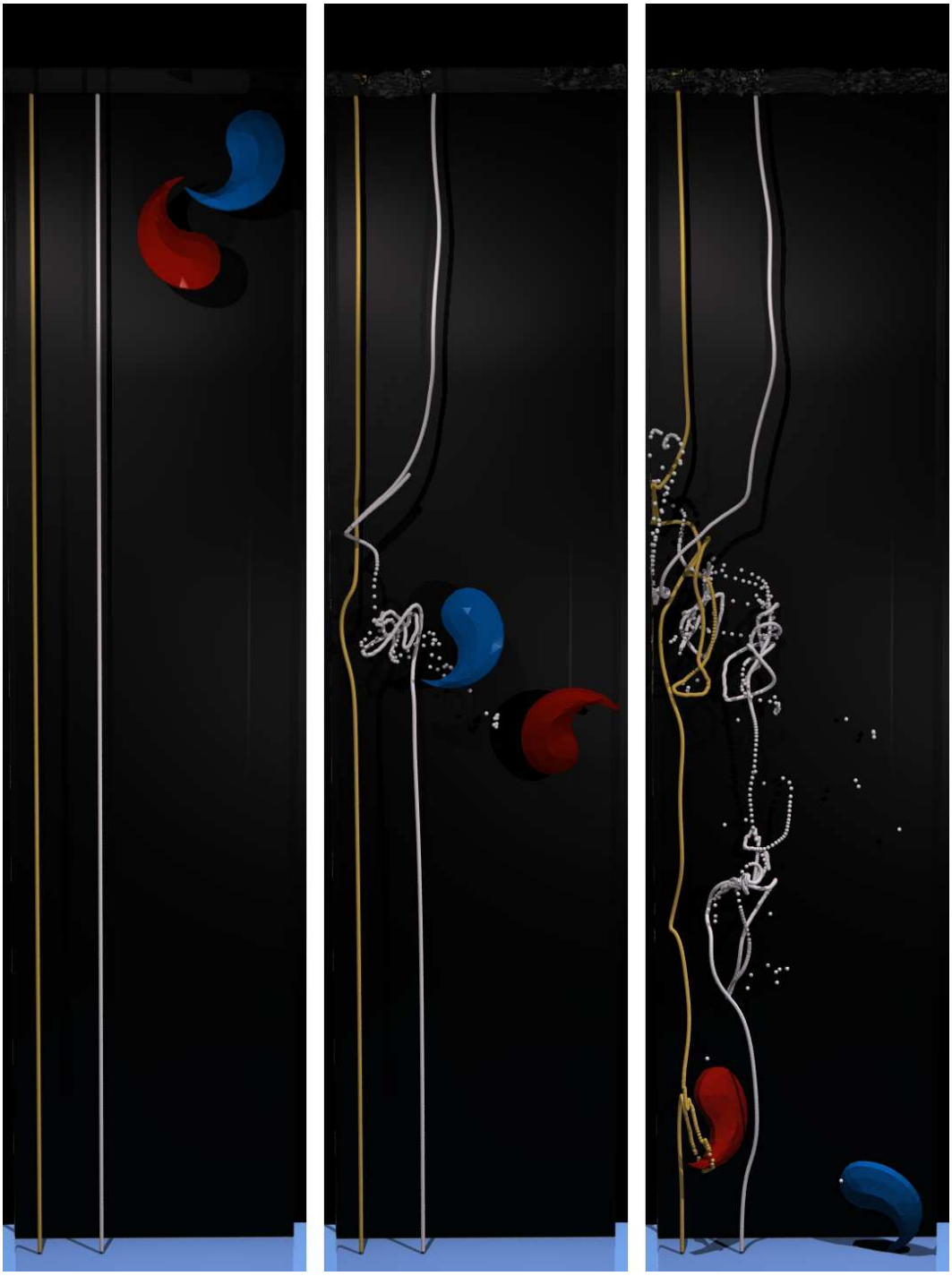

Figure 21: Two odd shaped gems tumbling in a water filled shaft.

similar drafting and tumbling for two odd shaped gems in the same virtual tank (figure 21). The detailed motion of these gems is very different from that of the spherical balls, because they have a preferred falling orientation.

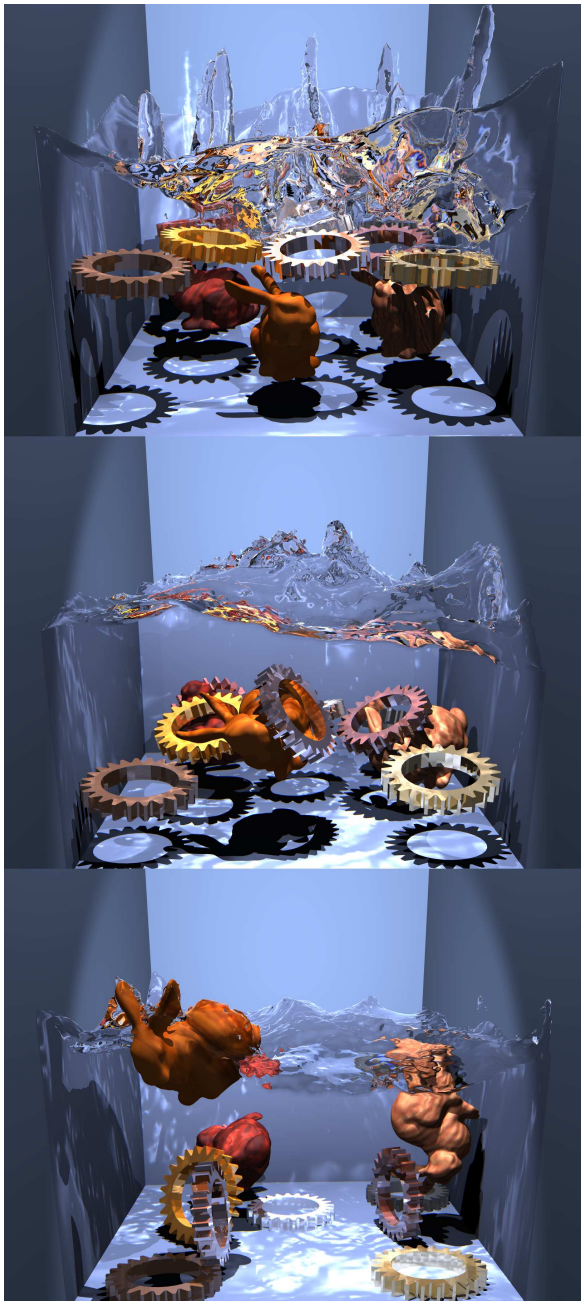

Figure 22: Eight metal gears are thrown down into a pool of water that contains wooden bunnies.

In figure 22, eight metal gears of relative density nine are dropped into a one meter wide tank of water. They hit the water at about 4m/s and make a rather large splash. In the water are three wooden bunnies. As the gears sink to the bottom of the tank they strike and interact with the bunnies which are, at the same time, trying to rise to the top. The bunnies eventually reach the surface, though the bunny on the right was lucky to escape the gear that hooked its ears.

Animation	Grid Size	CPU time	Steps	Coupling	Fluid	Level Set
Gears & Bunnies	$64 \times 68 \times 64$	2h 13m	430	5%	70%	25%
Optimized Grs & Bns	$64 \times 68 \times 64$	1h 9m	311	7%	60%	33%
Lead & Wood	$96 \times 64 \times 48$	5h 50m	1110	0%	65%	35%
Block Splash	$93 \times 73 \times 60$	2h 9m	635	28%	22%	50%
Tumbling Stones	$68 \times 292 \times 24$	14h 0m	609	0%	97%	1%
Tumbling Gems	$68 \times 292 \times 24$	21h 9m	605	38%	61%	1%

Table 8: CPU computation time for the first second of animation including the computational grid size, the total CPU time not including the render or file write time, the number of steps needed to reach the one second mark, and the percentage of time spent in functions that enable two-way coupling, solve the fluid equations and advance the level set.

5.7 Discussion

We have presented the rigid fluid method as a means of simulating many common two-way fluid-solid coupling scenarios. The main strength of the rigid fluid method lies in the efficient handling of the rigid solids, requiring only a relatively small additional computation on top of that already required by the fluid solver. This remarkably minimal cost follows from the use of distributed Lagrange multipliers [31] for the solid rigidity constraints, computed with an operator splitting that separately projects onto the divergence-free velocities in the entire domain and the deformation-free velocities inside the solids.

This remainder of this section lists some computation times; these times are further dissected and possible speed ups are suggested. I will then describe some interesting problems I have not yet solved for the method. Finally I will examine the strengths and the weaknesses of the rigid fluid method while describing possible future work.

5.7.1 Computation Time

Tables 8 and 9 demonstrate the speed of the rigid fluid method on a Pentium 4 2.8GHz with 2GB RAM. All simulations in [7] were re-run to collect more useful timing information. Table 8 shows computation times for the first second, when most of the action takes place, including total CPU time (not including render and file write time) and the number of simulation steps taken. Table 9 shows the computation times for the entire simulation. On all the times collected a 60fps animation rate was enforced, as well as a time step restriction that ensured no rigid body would move through

Animation	Seconds Simulated	CPU time	Steps	Coupling	Fluid	Level Set
Gears & Bunnies	8	8h 18m	1632	5%	75%	20%
Optimized Grs & Bns	8	4h 6m	1516	9%	74%	17%
Lead & Wood	8	13h 16m	2576	0%	71%	29%
Block Splash	5	9h 33m	2577	28%	26%	46%
Tumbling Stones	4	38h 58m	1621	0%	97%	1%
Tumbling Gems	4	51h 9m	1448	37%	62%	1%

Table 9: CPU computation time for the entire animation including the number of seconds simulated, the total CPU time not including the render or file write time, the number of steps needed to reach the end of the animation, and the percentage of time spent in functions that enable two-way coupling, solve the fluid equations and advance the level set.

more than one MAC cell at a time.[3] Two pressure projection steps were used instead of one for all but the Optimized Gears and Bunnies simulation, the exact optimizations used on that simulation are described in section 5.7.2. The particles, with an average density of 32 per cell, in the particle level set were reseeded every twentieth simulation step. For each of the simulations, there are three possible ways to decide if a point is inside or outside of an object, and the rigid bodies have differing levels of complexity. For the gears and bunnies simulation (figure 22) the 8 gears have 250 vertices each, and the 3 bunnies have 5002 vertices each, so distance fields were used to represent them allowing for constant time inside/outside tests. The two spheres in the lead and wood simulations (figures 2 and 19) and the tumbling stones animation (figure 20) have 2050 vertices, but a simple analytical function was used for constant time inside/outside tests. In the block splash animation (figure 17), and the tumbling gems animation (figure 21) a linear time inside/outside test is used. The linear test is a parity test that shoots a ray out to infinity and counts the number of intersections the ray has with the triangles of the rigid body; it is an $O(T)$ test, where T is the number of triangles in the blocks ($T = 12$) and the gems ($T = 340$).

5.7.2 Possible Speedups

I will break the computational speedups into three categories: speedup of the two-way coupling, speedup of the Navier-Stokes simulator, and speedup of the level set representing fluids surface.

[3] This restriction is sort of a CFL condition on the movement of the rigid bodies. The strictness of this condition is painfully apparent in the Lead & Wood animation because the spheres were rotating very fast and we take maximum rotational speed into account. We have not attempted to remove this restriction, but we believe it can be relaxed considerably.

Each of these sections are discussed in turn, next.

Almost all of the time spent in the functions that create the two-way coupling is from the inside/outside test for the rigid body. After advancing the computational domain, $\mathbb{C}$, a list of MAC cells that contain part of a solid must be created. Simple axis-aligned bounding box and bounding sphere tests allow me to ignore cells that are far from the solid. Once a point is inside the bounding sphere and box, however, the point must pass a more rigorous test. It is clear from tables 8 and 9, that the constant time analytical test used in the Lead & Wood and Tumbling Stones animations are the fastest, and that the linear time parity test used in the Block Splash and Tumbling Gems animations take the longest. Exact analytical formulae are not usually available for complex shapes, but improving the speed of the static signed distance test, used in the Gears & Bunnies animation, could provide comparable speeds. Aside from speeding up the test themselves, one could reduce the number of tests needed. The current rigid fluid implementation will test up to 64 points in a cell as described in section 5.5, but I believe only one test per cell corner is necessary. Once the distance is known at each cell corner, a closed polygonal object could be created for that cell and a good guess for the cells center of mass and moment of inertia could be obtained from it [51]. I did not implement this possible speedup, but it should work for most objects unless they have several small resolution features.

Stam observed that the semi-Lagrangian technique used for advection is more accurate if it is used on a divergence free velocity field [83]. Because of this observation we decided to solve the Navier-Stokes equations in the following order:

1. Add Body Force,
2. Viscous Diffusion,
3. Pressure Projection,
4. Advection,
5. Pressure Projection.

However, with the two pressure projections steps, most of the time spent solving the fluid equations is spent solving for pressure: 94% for Gears & Bunnies, 90% for Lead & Wood, 73% for Block

Splash, 96% for Tumbling Stones, and 97% for Tumbling Gems. The optimized loop order I used in the optimized Gears & Bunnies animation is:

1. Advection,
2. Add Body Force,
3. Viscous Diffusion,
4. Pressure Projection.

The optimized loop seems to give water movement that is slightly less lively for most simulations, however, comparable results are obtained for most simulations. The exception is the Lead & Wood animation, where the dampening practically destroys the movement on the water's surface and no high splashes are obtained. Using a single pressure projection can nearly double the speed of the the fluid solver, but I wold recommend using a more accurate formulation for advection than the semi-Lagrangian one.

A 5^{th}-order accurate in space WENO method was used to advance the level set representing the fluid free surface, along with a 3^{rd}-order accurate in time TVD-RK method (see section 6.2 for details). Even using the high order accurate technique, nearly 75% of the time spent in the particle level set was spent with the particles. The particles are not needed if there are no splashes or high curvature areas, so for the optimized Gears & Bunny simulation we removed all the particles from the simulation after the first second. The particles level set technique is discussed in more detail in chapter 6.

With these simple optimizations, the Gears & Bunnies animation goes from taking 62 seconds on average to simulate one animation frame at 60fps to taking only 28 seconds per frame, as can be seen by tables 8 and 9. The results of the optimized gears and bunny simulation can be seen in the right hand column of figure 32. The smoothness of the fluid surface, however, is due to the treatment of the fluid surface as discussed in section 6.3, and not the speedups discussed here.

5.7.3 Ulrabuoyant and Wet Rigid Bodies

There is an instability in the momentum source term, **S** (equation 70), when dealing with the momentum of ultrabuoyant rigid bodies. We have found that **S** term is fine for objects ranging from

infinitely heavy to the lightness of wood (about 0.45 relative density). The unstable term in **S** is

$$\mathbf{S}^{unstable} = (\rho_r - \rho_f)[\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t}]. \tag{84}$$

The nature of the instability stems from two places in the above equation. The first is that when the liquid is much denser than the solid the factor, $(\rho_r - \rho_f)$, gets larger in magnitude. This, in itself is not really a problem, after all if $\rho_r \gg \rho_f$ then the magnitude can be large as well. The first issue manifests in equation 71 when **S** is divided through by ρ_r. When ρ_r is large it will balance out the large magnitude term above and keep the terms magnitude less than one. However, if $\rho_r \ll \rho_f$ then the term approaches infinity as ρ_r gets smaller. Normally we could fix this with a fractional step procedure by reducing the time step. Unfortunately, this brings up the other instability issue with the term. Equation 71 also multiplies **S** by Δt and in so doing, cancels the Δt in the denominator of equation 84. Therefore changing the time step does nothing to change the instability.

The good news is that simply removing the unstable term when simulating ultrabuoyant rigid bodies gives visually pleasing results, even though it does not, strictly speaking, conserve momentum. There is another problem with ultrabuoyant objects besides just the the instability, however, and simulating a styrofoam object will expose it. If a submerged object is touching water and floating on top of it, how does the object know it is no longer supposed to be wet? If the rigid body is light enough, then the smallest drop of "dense" water stuck on or in the object will make the object rise. We created several entertaining simulations of styrofoam balls bursting out of deep water and flying to the roof of the computational domain, where they would stay while emitting little droplets of water. So, to simulate ultrabuoyant objects one needs to be aware of the instability, and also be vary careful to keep their rigid bodies dry.

5.7.4 Future Work

Several fruitful avenues for future work remain. It is possible to do both the divergence-free projection and the rigidity projection in the same step, but the computational cost is substantially higher than doing either projection alone. There are, however, environments that need both projections done at the same time. For example, if the top half of an hour-glass was filled with water and a rigid ball were to plug the hole separating the top and bottom reservoirs, then both projections and the rigid body contact constraints should be done at the same time or water will seep through the

plug. If these types of situations are common in a given scene, then we recommend trying an ALE method, or a finite element version of the DLM method [31].

One simple addition to the method as presented here would be to add joint constraints for our rigid bodies. The α_c and $\mathbf{A}_c$ variables do not need to come from collision and contact forces, they could be used to model the forces necessary for joints, or even human motion controllers. We would like to add these features to our rigid body solver so that simulated divers [102] could splash and interact with the water, and maybe learn to swim.

As mentioned in section 5.7.3, we have not done much to improve the surface of the particle level set where the water and the solid objects meet. At the moment, we simply remove particles that find their way into rigid objects, but beads of water still seem to stick to objects that have just popped out of the water. We are looking into ways to visually improve this. We do note that using a level set without particles gives us a nice separation of solid and water when we want it, but then thin splashes of water loose too much volume.

The difficulties of dealing with the surface where it meets the object are exacerbated if the scale of the simulation is small enough; in such cases, static and dynamic contact angles would need to be properly maintained between the fluid and solid surfaces. The length scales of the present animations, in contrast, are so large as to safely ignore contact-angle effects, since the capillary length of the air-water surface is on the scale of millimeters. Cohen and Molemaker [12] have sketched out some research that could help in this area.

Objects that occupy very few grid cells are difficult to simulate. A plank of wood is fine as long as it always takes up at least one grid cell along its length, but extremely thin rigid objects, like the wall of a metal bucket, can not be simulated without a sufficiently fine grid, or else the rigidity constraint will not stop water from flowing through the thin walls of the bucket.

We would like to simulate large scale phenomena like a battle ship sinking, and still capture high frequency motion. A multi-resolution solver would allow for a larger computational domain. We believe our rigid fluid technique can be adapted to a multi-resolution fluid simulator similar to [76].

Objects thrown into sticky liquid, like honey, will move differently than objects thrown into slippery liquids, like oil. We have given an equation for the traction force around the object, but have not experimented with it. There are several affects that could be added with improved treatment

of the traction force and solid/liquid boundary treatment.

CHAPTER VI

MISCELLANEOUS FREE SURFACE ISSUES

"BOUNDARY CONDITIONS AT THE FREE SURFACE REMAIN THE MOST INTERESTING OF THEM ALL. FOR THE NUMERICAL METHOD, THEY ARE THE MOST DIFFICULT CONDITIONS TO APPLY ACCURATELY."

Francis H. Harlow [99]

Figure 23: Tearing free surface splash.

Francis Harlow wrote the words quoted above in 1964 in his chapter of the original Marker-And-Cell technical report [99]. It amazes me how his words still ring true forty years later. Much of today's cutting edge research deals with what I will call *free surface* issues.[1] These issues are not just boundary conditions; they include the basic representation of the surface, tracking its movement, determining how it moves, and of course visualizing it. This chapter could have been spread throughout the chapters of this work, but I believe free surface issues are important enough to warrant their own chapter. I have come to believe, and been known to say, that simulation would be easy if it weren't for the boundary conditions. All the difficulties are in the boundary conditions,

[1] In the CFD literature a free-surface flow is a fluid simulation that assumes that atmosphere exerts no shear stress or inertia on the fluid.

and the free surface is indeed the most interesting of these difficulties. This chapter covers a few odds and ends I learned when dealing with the free surface issues.

6.1 Separating Liquid From Empty Air

Chapters 3 and 4 used massless marker particles to delineate the fluid from the empty air, and chapter 5 used a particle level set. This section will discuss the pros and cons of using just marker particles, and touch briefly on what advantages a level set has over using marker particles alone.

The biggest advantage of using marker particles alone is simplicity. When deciding if a MAC grid cell is fluid or air, one only needs to know whether a cell contains a marker. If it does, then it is fluid. The markers also give a convenient way to *carry* information into empty cells from the pervious time step; in that sense they take care of the advection boundary conditions in a simple manner. Another advantage is the detailed high frequency information about the shape of the simulated fluid that the particles carry. Consider the coarse 16 by 16 grid depicted in figure 24. The particles in the grid obviously hold much more information about the surface than the grid itself.

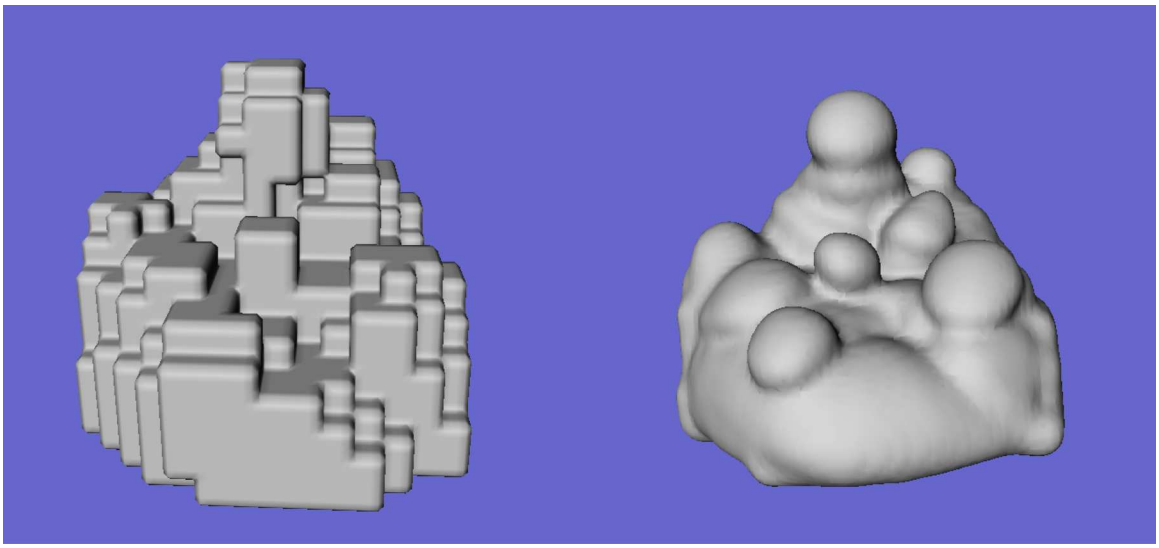

Figure 24: Grid resolution (left) compared to particle resolution (right).

The fluid simulator in chapters 3 and 4 use only marker particles to track the surface, but the high resolution information of those particles is not available to the fluid simulator. Thus even basic geometric information about the surface, *i.e.*, surface normal or curvature, cannot be used. In fact, extracting the triangulated surfaces to be rendered for that work was an off-line process. The

pipeline depicted in figure 25 shows the steps taken to create the final image. First, all the particle locations are written to a file, then a splatting technique is used to create a volume in a grid with double the resolution of the simulation grid. With more particles, an even higher resolution grid could be used. After the triangles are extracted the surface was rendered (with subsurface scattering in the case of the wax bunny). The surface splatting technique was created and implemented by Greg Turk, and the subsurface rendering technique was created in [43] and implemented by Brooks Van Horn; details can be found in [8].

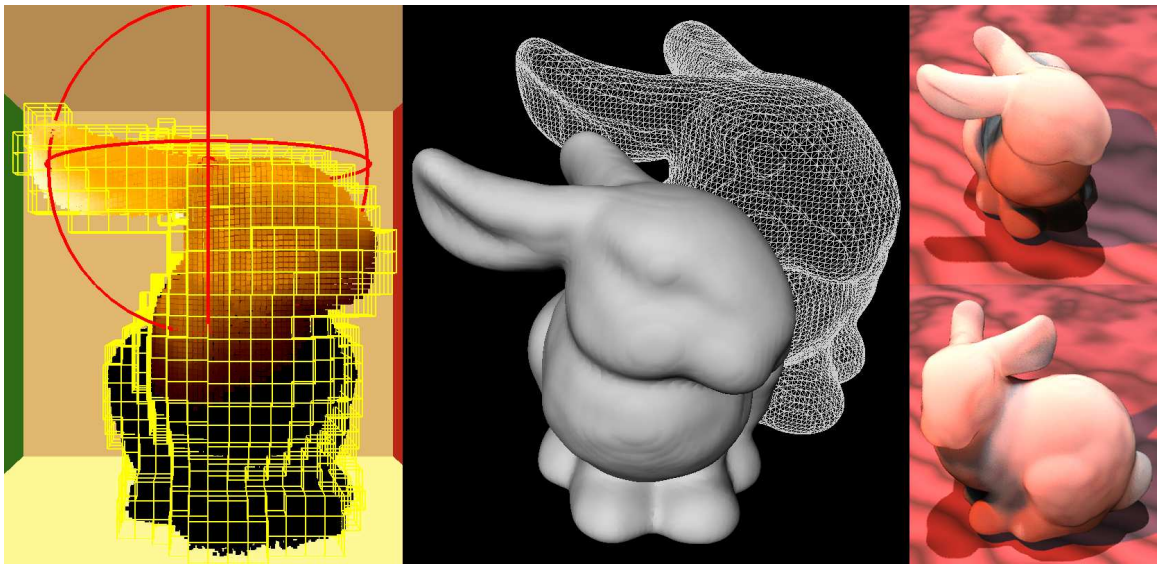

Figure 25: A bunny from particles to triangles to final render.

Considering the coarse resolution of the grid for the bunny simulation, $35 \times 28 \times 38$, we were impressed with the detail of the surface that the marker particle alone afforded us. However, another representation for the free surface, known as a *level set* [57], was gaining popularity in the computer graphics literature [16, 24]. A level set is a dynamic implicit surface. The term *level set* is a mathematical term that simply means the curve in 2D or surface in 3D of a function at a specific value. For example, the 0-level set of the 2D implicit function $x^2 + y^2 - 1 = 0$ is simply a circle with radius one, and the 2-level set of the of 3D implicit function $x^2 + y^2 + z^2 - 1 = 0$ is a spherical shell with a radius of three. Over time, the term level sets has changed to the study of these surfaces as they are manipulated with various partial differential equations and functions [73, 58].

As mentioned above, the marker particles alone gave us a nice surface, but the surface was hard

to extract from the particles. The simulation times in table 6 are relatively fast at a couple frames per second, but the surface extraction for those sequences were not so fast, taking the popular "overnight" time span. Aside from the popularity of the level set technique, there were several reasons we chose to use it for the free surface representation of the rigid fluid technique detailed in chapter 5. The unacceptably long extraction times of the surface was the first reason we decided to move to a level set technique, because a fast marching cubes algorithm can be used to extract surfaces at interactive rates with level sets (see section 6.4). Another reason was that we could get curvature, and normals from the surface. Some issues with level sets are discussed in detail in the next section, and hopefully my nearly three year love/hate relationship with them can give future graphics researchers some insight.

6.2 Level Set Issues

Level sets are used in chapter 5 for extending the velocity of the fluid out into the air (details in section 6.3), quick triangle surface extraction (details in section 6.4), and defining and tracking the fluid free surface. This section discusses details of our level set implementation.

6.2.1 Important Level Set Properties

We view a level set as just another scalar field (ϕ), like the temperature (t) or pressure (p), with special properties. The most important region of the level set for us is where it equals zero (green in figure 26). The 0-level set represents our free surface, so many times we will refer to the level set and really mean where $\phi = 0$. Because the free surface is the most important part of the level set, we only need to pay attention to a *narrow band* [1] of cells around it. We initialize the band around the free surface at every step, so our narrow band method is more accurately known as a *sparse-field* method [100]. Figure 26 shows the popular Zalesak's Disk [106], which will be used to demonstrate some concepts throughout this section.

All points inside the water (blue in figure 26) have $\phi < 0$ and all points outside of the water (orange in figure 26) have $\phi > 0$. Knowing the above, the outward normal of the free surface, $\mathbf{n}_\phi$, is:

$$\mathbf{n}_\phi = \frac{\nabla\phi}{|\nabla\phi|}. \tag{85}$$

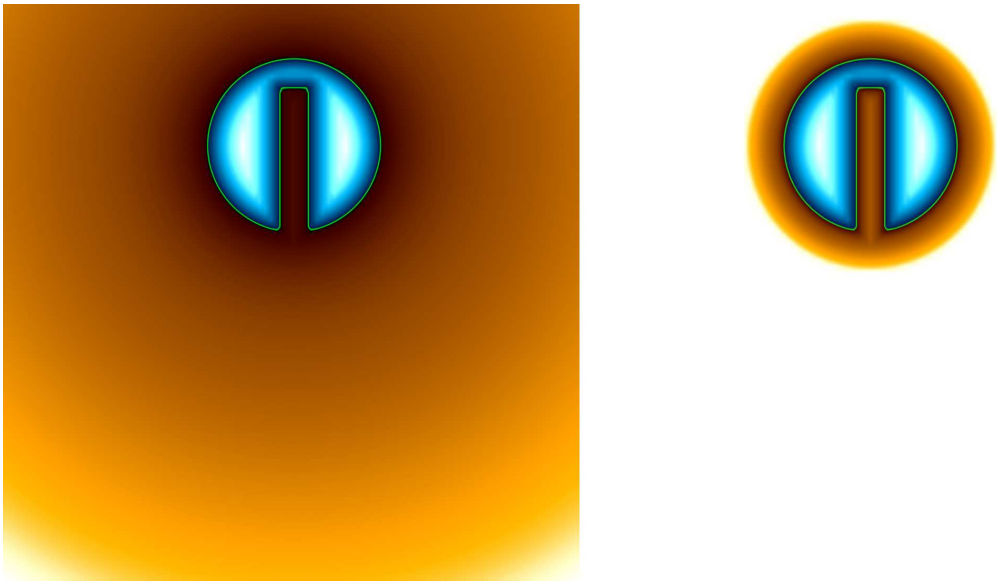

Figure 26: Zalesak's disk (left) and the sparse-field version (right).

Another important property of ϕ is that it is a signed distance field,[2] that is to say that no matter where we take the value of ϕ, we know that the free surface is a distance $|\phi|$ from that point, and that $|\nabla\phi| = 1$. This fact allows us to ignore the denominator in the above equation and simplify it to:

$$\mathbf{n}_\phi = \nabla\phi, \tag{86}$$

and therefore skip the costly computation of $|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}$. In fact, when ϕ is a signed distance function, we replace $|\nabla\phi|$ with 1.

6.2.2 Reinitialization

The signed distance property is maintained through a process known as *reinitialization*. One popular way to reinitialize the level set is with the fast marching method [74], which is an $O(N \log N)$ technique based on sorting (the reason it is $O(N \log N)$). We will describe an $O(N)$ PDE based scheme, but first we must introduce upwinding now because it is important to understand for the solution technique we chose for reinitialization. In our level set implementation we use a grid spacing of one, because it simplifies the equations considerably. Consider two ways that we can take a first derivative of ϕ in the x-direction: the forward difference operator, ϕ_x^+, and backward difference

[2] Heavy-side and other functions [74] can be used to represent a level set, but the signed distance field is important for our implementation.

operator, ϕ_x^-, at i, j, k are

$$\phi_x^+ = \phi_{i+1,j,k} - \phi_{i,j,k}, \tag{87}$$

and

$$\phi_x^- = \phi_{i,j,k} - \phi_{i-1,j,k}. \tag{88}$$

The idea behind upwinding is that information can only travel in one direction for some functions, and based on a characteristic function, **w**, one should use either the forward or backward difference version of the derivative (*not* the same as the forward and backward Euler derivatives in section 3.2.2).

The upwind version of the reinitialization function that we use is:

$$\phi_t + \mathbf{w} \cdot \nabla\phi = S(\phi), \tag{89}$$

where the characteristic function, **w**, is the outward normal of the level set from equation 85 instead of equation 86 because we can not assume the signed distance property on ϕ while reinitializing it, and S is the signature function. S is much like the sign function, but has $S(0) = 0$ and is dependant on the reinitialization implementation that us used.

During the reinitialization, we wish to preserve the high frequency (sharp corners) spatial information of the level set. For this reason we use a 5^{th}-order WENO scheme to discretize the spatial derivatives in equation 89. Details of our WENO scheme can be found in [58, 17, 77], and replace the first order upwind derivatives in equations 87 and 88. WENO stands for *weighted essentially non-oscillatory*. It is named weighted because different combinations, weights, of 3^{rd}-order ENO stencils are used to create a 5^{th}-order scheme away from sharp corners. Consequently, WENO is only 3^{rd}-order in areas of high curvature. It is named essentially non-oscillatory because the high frequency information that may be lost in one step can be regained in future steps [3].

The time derivatives in equation 89 are computed with the 3^{rd}-order *total variation diminishing* (TVD) Runge-Kutta scheme found in [58, 17, 77]. TVD schemes are meant to control the oscillations in numerical schemes by preventing values at step $n+1$ from exceeding the range of values at time step n [3]. Because the reinitialization's job is to smooth the level set, not update it's position, the loss of high frequency information generally associated with TVD schemes [3] does not seem to be a problem.

With all that background out of the way, we can now get back to the point at hand–the technique that we use for reinitialization. For our level set implementation we chose a PDE-based reinitialization [63] instead of a fast marching method for two reasons: the PDE-based method is is $O(N)$ instead of $O(N \log N)$, so it should scale better to larger grid sizes, and the PDE-based reinitialization allows us to use the higher order WENO and TDV-RK techniques. Fast marching methods are notoriously difficult to extend to higher order. For most calculations we have a band width (β in [63]) of 4 voxels and a smooth band width (γ in [63]) of 7 voxels, though these variables can change based on our needs. It should also be noted that the PDE-based method we use does not move the level set across cell nodes as the technique in [86] does, but we use a similar test as [86] for deciding when smoothing is complete, namely:

$$\frac{\sum_{|\phi^n_{i,j,k}| \in \text{band}} |\phi^{n+1}_{i,j,k} - \phi^n_{i,j,k}|}{M} < \varepsilon \Delta\tau, \tag{90}$$

where $M \equiv$number of grid points in the narrow band. In our implementation we found a value of $\Delta\tau = 0.5$ and $\varepsilon = 0.1$ works well. Our smoothing almost always takes only one step, and recall that the grid spacing is always 1.

6.2.3 Moving the Level Set

So far we have talked about only using the cell points around $\phi = 0$ that are necessary, and we have talked about reinitialization of the level set to a signed distance function, but level sets are not useful to us unless we can use them to track the movement of the free surface. Therefore, we need techniques that can move the level set based on an external velocity field that we supply. In the case of fluid simulations this velocity will come from the MAC grid (section 3.2.1), but for now we will consider an artificial velocity field, rotating counter clockwise as depicted in Figure 27.

The *level set equation* used to move a level set with an external velocity field is:

$$\phi_t + \mathbf{u} \cdot \nabla\phi = 0. \tag{91}$$

The vector field, $\mathbf{u}$, is the external velocity field and can be thought of as the fluid velocity field when we use the level set to track free surface. There are many ways to discretize equation 91 from semi-Lagrangian, to WENO. When upwinding techniques like the WENO method is used then the characteristic function is the velocity (*i.e.,* $\mathbf{w} = \mathbf{u}$). Figure 28 shows what happens when Zalesak's

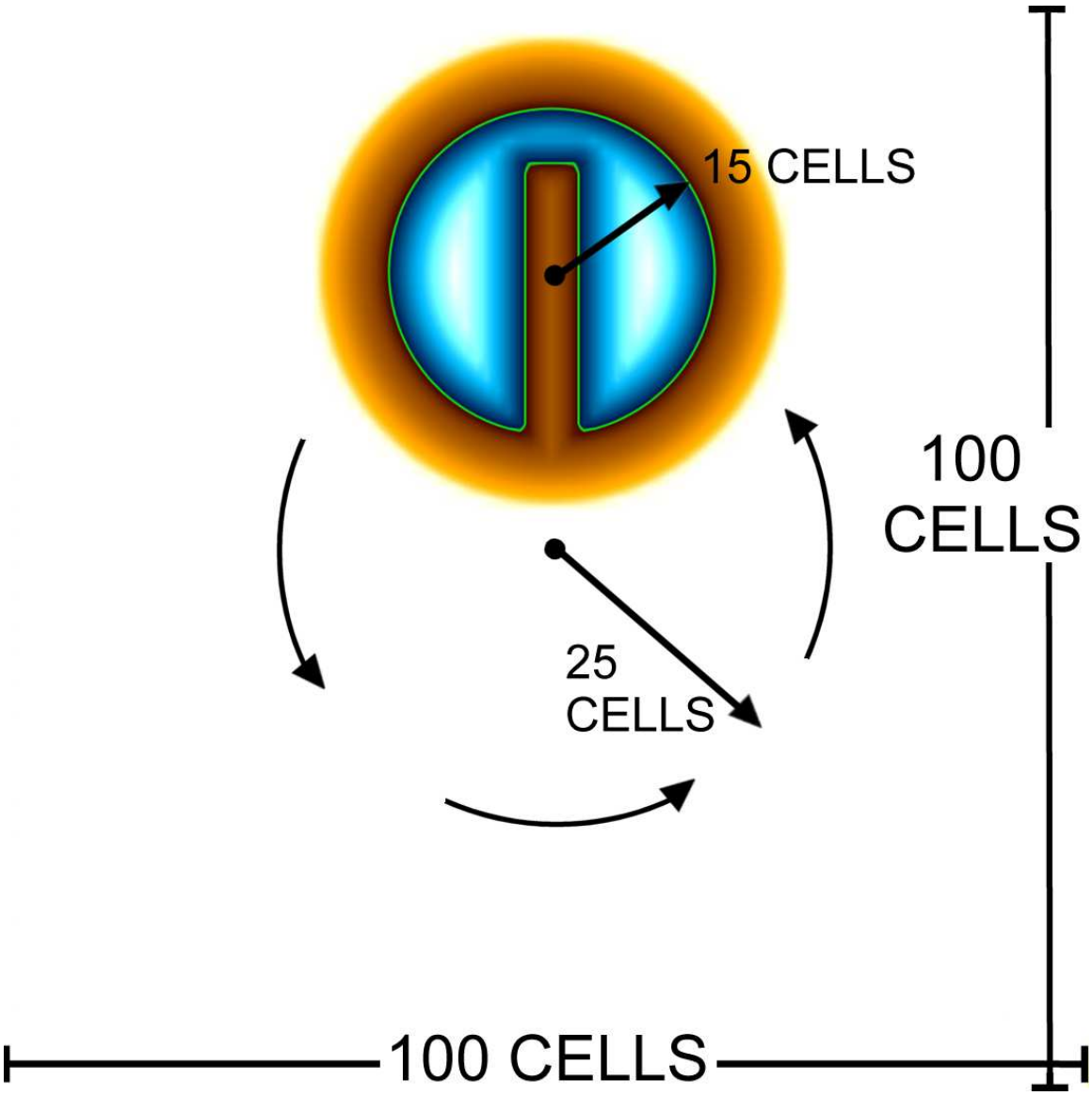

Figure 27: Schematic of Zalesak's disk [106] which traverses one complete solid body rotation about the center of the computational grid in 628 time steps.

disk is rotated 360^{o} using various discretizations. At the end of the rotation we want the disk to look the same as when it started. Because of the odd array of shapes we created in Figure 28, it is obvious that we need more than just a good discretization of the level set equation to get good volume (area in 2D) conservation.

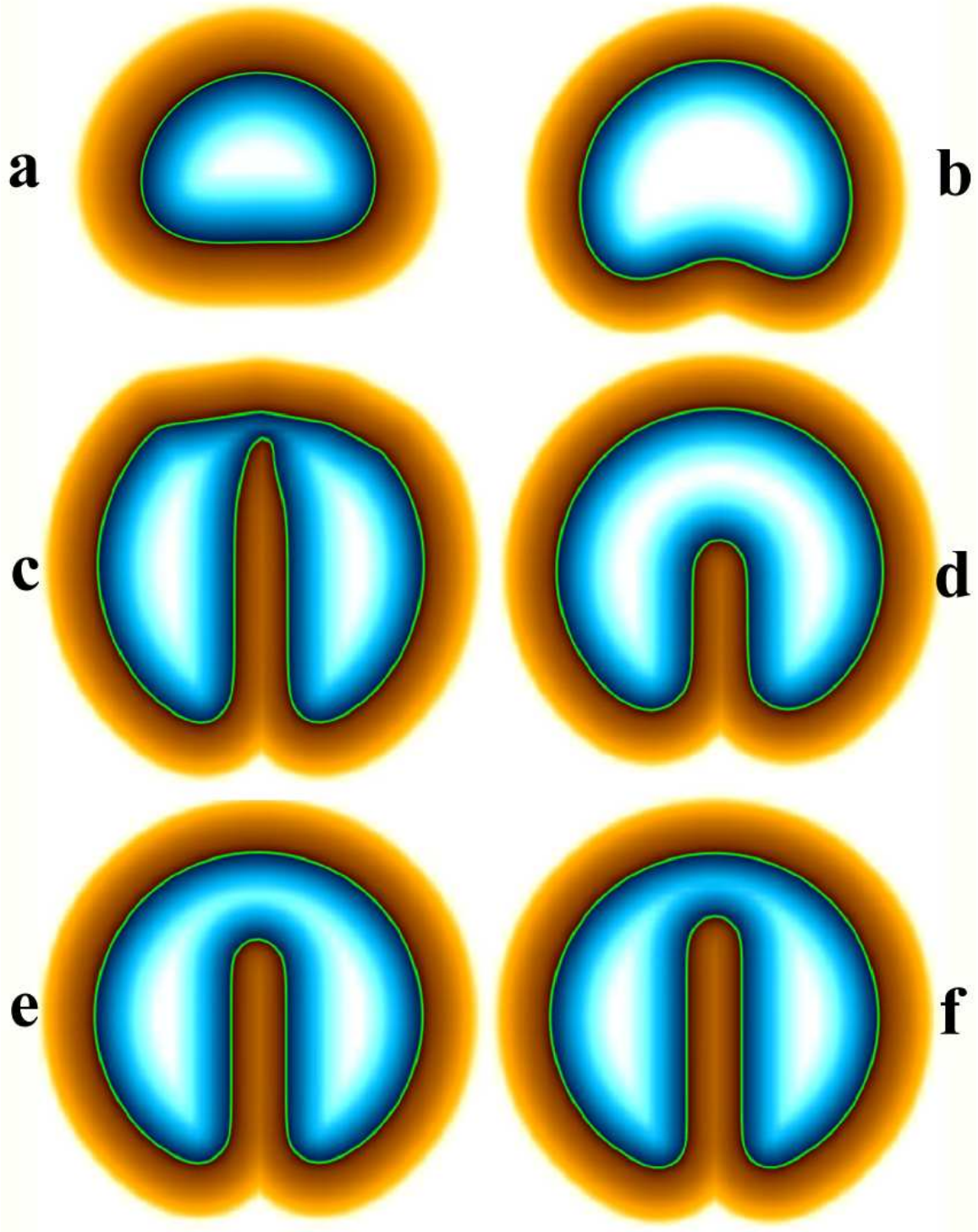

Figure 28: Zalesak's disk after rotating 360^{o} using various discretizations of equation 91: semi-Lagrangian (a), forward Euler with 1st-order upwinding (b), forward Euler with 2nd-order ENO (c), 2nd-order TVD-RK with 2nd-order ENO (d), 2nd-order TVD-RK with ENO (e), and 3rd-order TVD-RK with WENO (f).

Figure 29: Lagrangian particles are placed on both sides of the level set in the particle level set method.

6.2.4 Particle Level Set

Part of the problem with tracking the free surface on a discrete grid is that we can only get viscosity solutions to the level set equation. That is, even if we use the 3^{rd}-order in time TDV-RK technique and the 5^{th}-order in space WENO technique, there will still be 1^{st}-order errors due to the coarse grid resolution. The *particle level set* [14] was created to compensate for some of the inaccuracies when advecting level sets by adding Lagrangian particles to the simulation and letting them correct the first order error with the inherent high resolution detail near the front that the particles contain.

Figure 29 shows the initial particle placement in Zalesak's disk before the rotation. Figure 30

shows that it does not matter a whole lot which advection technique is used if the level set is corrected with particles. The level set advection experiments in this section were performed in September of 2003, but the same conclusion was independently observed in [15].

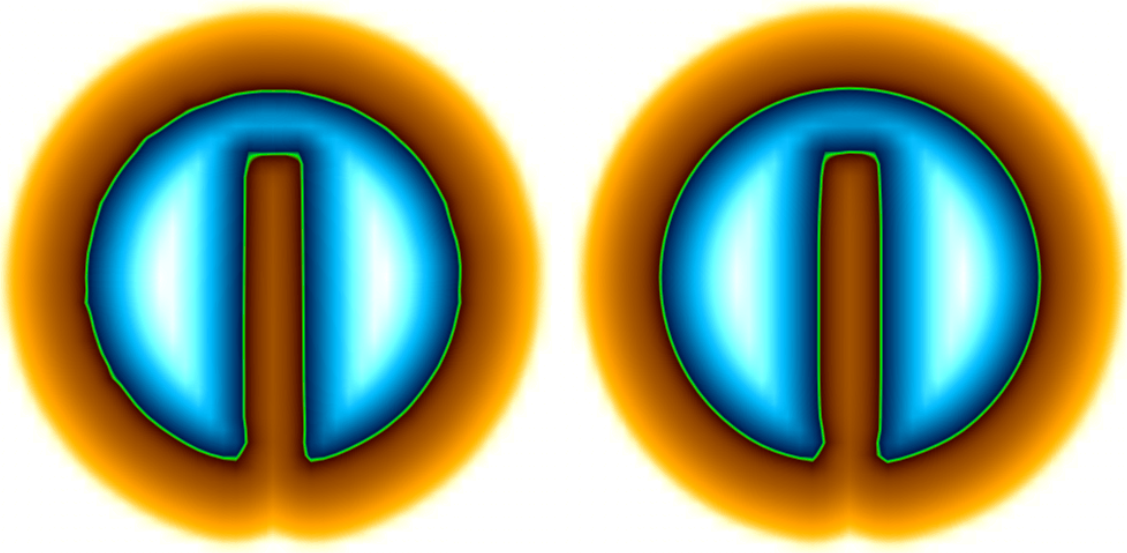

Figure 30: Zalesak's disk rotated 360^o using the particle level set technique: semi-Lagrangian (left), and 3rd-order TVD-RK with WENO (right).

Though only 2nd-order accuracy can be gained for particle advancement when linear interpolation is used for the velocities, we use 4th-order Runge-Kutta because it is less time-step restrictive (*i.e.*, has a larger stability region [93]) and allows us to change our interpolation scheme to higher order and to gain accuracy later if we wish.

6.3 Velocity in the Air

The velocities in the empty atmosphere are used to move the positive particles in the particle level set. The velocities also allow us to use higher order spatial derivatives for the inside of the fluid because they create enough boundary values for use with larger finite difference stencils. This section will cover two issues with the velocities that are in the empty atmosphere outside the fluid. This section first expands on the issue of continuity conditions in section 3.2.3.4, and then describes the technique we use to extend velocities several cells out into the empty air.

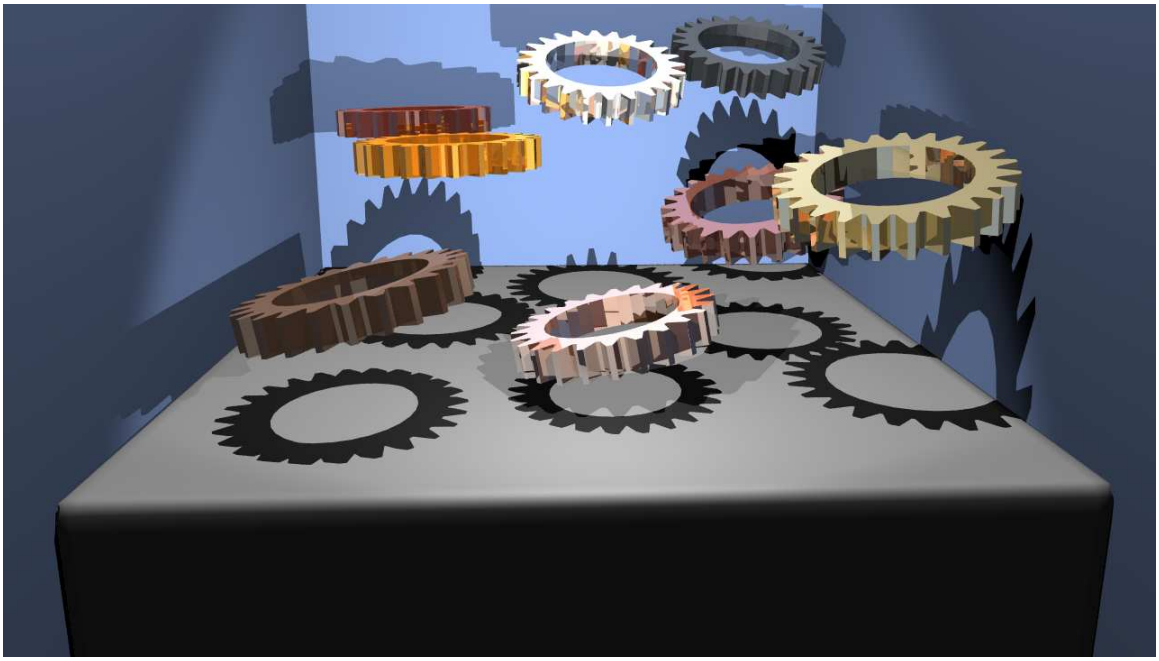

Figure 31: The first frame from the Gears and Bunnies simulation with opaque water surface.

6.3.1 Velocity at Surface Cell Boundaries

The simulations in chapter 5 have a nice lively active fluid surface, but near the end of there simulations there are some problems. We will be discussing these issues using the Gears & Bunnies simulation depicted in figure 22 as an example. Frames from this animation will be re-rendered in this section so that the surface of the water is easier to see, like figure 31 which shows the first frame of the animation–take note of the water level. Figure 32 shows a comparison of two simulations run with identical starting conditions, but the simulation loop and free surface treatment are different. The left hand side of figure 32 was run with the loop detailed in section 5.4, and the right hand side uses the optimized loop already described in section 5.7.2. The big differences between the loops are that the optimized loop solves one pressure projection step instead of two, and the particles in the optimized version are removed after one second. The different free surface treatments for the two loops are described next.

The surface shown in the left column of figure 32 is noticeable noisier than the surface on the right; this noise is a problem, and is glaringly evident, when the surface of the water should be smooth like in the last frame of the animation. Notice, also, that the fluid on the left has lost volume by the last frame. However, even though the surface on the right is smoother and has better volume

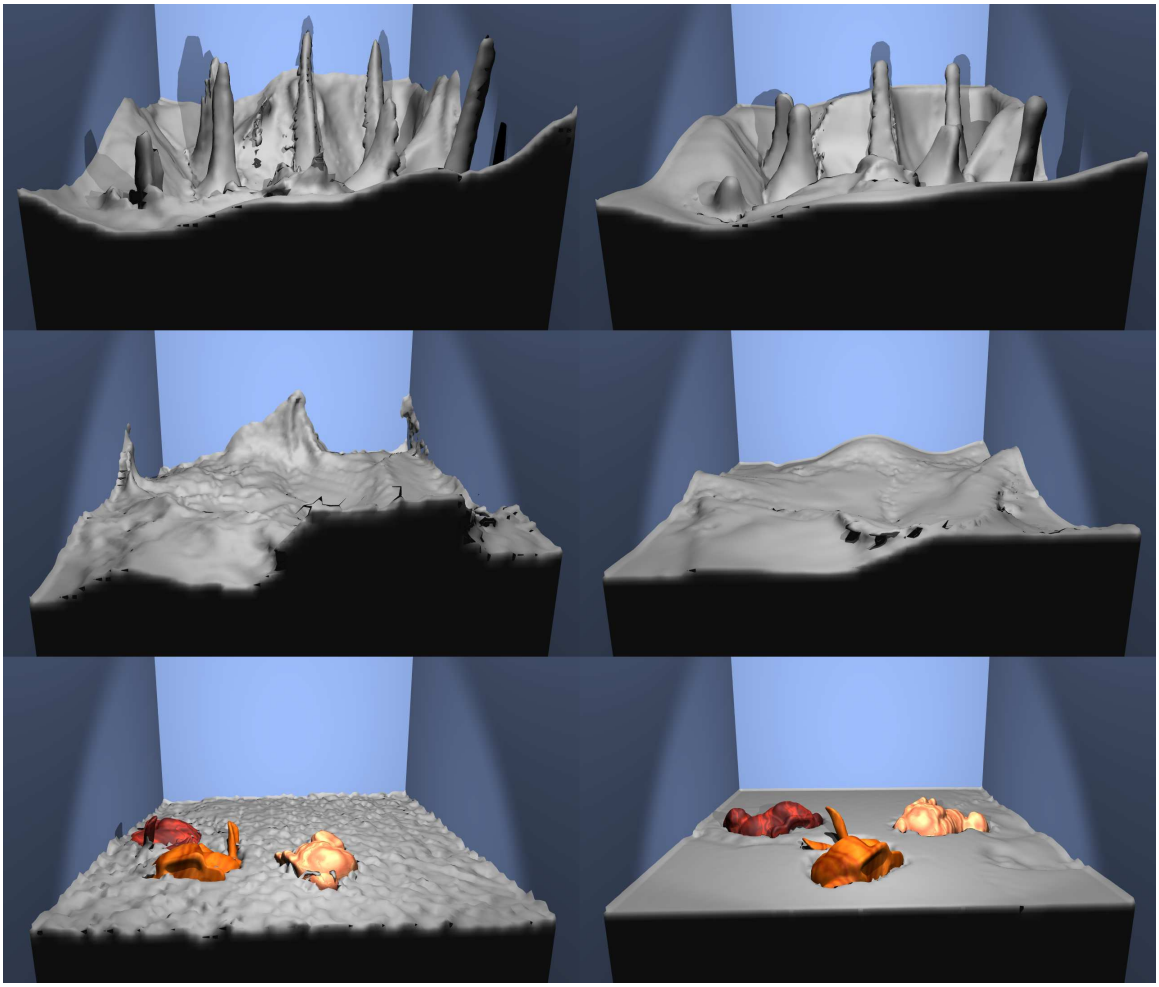

Figure 32: Comparison of the fluid surfaces for two different simulation loops of the Gears & Bunnies simulation after 0.33*s* (top), 1*s* (middle), and 8*s* (bottom). The left column is the same simulation loop used in all simulations pictured in chapter 5, and the right column is the optimized (and smoother) loop described here and in section 5.7.2.

preservation, it does have some trade-offs compared to the surface on the right. The top of figure 32 shows the point in the animation where the gears have just splashed into the water and the small pillars of water that gush up through the center of the gear are at their highest point. Notice that the gushes on the left reach higher into the air. Though is is difficult to show with still frames, the water on the left remains lively much longer than the water on the right. The center frames were taken when the splashing up the walls of the simulated pool are highest; the splashes on the left are obviously more vibrant. In fact, the fluid in the left simulation remains lively–sloshing back and forth–for most of the simulation, while the animation on the right dampens out as if the fluid was more viscous than water. Animators will have to choose if the deficiencies in the noisy animation

are worth having a dampened animation. The issues discussed here (surface noise, volume loss, and artificial surface dampening) are not discussed in detail in the graphics community. I will detail the two treatments of the surface with enough detail so readers can, hopefully, reproduce my results. Note however that there are still many avenues to explore and a lot of trial and error was used to create these two loops.

The continuity equation (equation 1), is enforced for each surface-cell[3] before each pressure projection. The enforcement of this divergence free condition is the same for both the noisy and the smooth surfaces, and is described in detail in section 3.2.3.4. The first difference is in the pressure projection step (section 3.2.4). When setting up the pressure matrix for the noisy simulation the surface-cells have their pressures set to zero, and these surface-cells are not included in the matrix at all; this decision to leave the the surface-cells out of the pressure projection matrix is, I believe, what causes the volume loss in the noisy simulation. When setting up the pressure projection for the smooth simulation, zero Derichlet boundary conditions were set for all the empty air cells and the surface-cells were included in the matrix along with all other fluid cells.

The other difference between the smooth and noisy simulations is the way that the air cells are given velocities once the free surface position has been updated. At the beginning of the simulation loop, both the noisy and smooth simulation level sets get the cell-centered velocities from the previous steps velocity as described in section 5.5. Once the velocities from the fluid are passed to the level set, an extension velocity (described next in section 6.3.2) of this is grown from the cell-centered (averaged) velocities into the cell centers that are in empty air. The level set is then moved into a new position. The differences are in how the surface velocities of the updated fluid domain position (obtained from the new level set position) are computed, and in what fluid velocities are actually updated. Before the fluid position is updated, a collection of cell faces that need new velocities is created. For the noisy simulation, the cell faces that need new velocities are the empty-faces that are near the fluid surface. In the smooth simulation, the cell faces that need new velocities are both the empty-faces and the surface-faces that are near the fluid surface. After collecting the cell faces that need new velocities from the old fluid domain position, the fluid position is updated my moving the level set as described above. Once the position of the fluid domain is updated the new

[3]Remember a surface-cell is a MAC grid cell that contains fluid but has at least one face with empty air on one side.

velocities are computed. In the noisy simulation, these new fluid velocities are obtained by linearly interpolating and scaling the cell-centered level set extension velocities. In the smooth simulation the new velocities are obtained by running new extension velocity equations for each of the separate components of the velocity that are stored at the staggered grid positions. Although finding out what faces must be updated requires looking at the old position of the fluid, the normal in the extension velocity equation (equation 92) is from the updated level set position.

6.3.2 Velocity in the Empty Atmosphere

The equation for extending the velocity of the fluid into the empty atmosphere is a rather simple one compared to many of the equations in this thesis, and is very similar to equation 89. We will consider the equation for extending the scalar quantity u, and the equation for the other components of velocity are nearly identical. The equation is

$$u_t + \mathbf{n}_\phi \cdot \nabla u = 0, \tag{92}$$

where $\mathbf{n}_\phi$ is obtained from equation 86. This equation simply extends the velocities out from the surface so that the velocity along a path following the normal away from the surface will always be the same along that path. Note that the positions of the velocities, whether at the cell centers or at the individual cell faces, are not not important as long as a reliable normal can be obtained for that position.

To extend each component of the velocity (or any scalar quantity for that matter) into the air, we solve a few iterations of equation 92. We use a standard upwind discretization of the spatial derivatives (equations 87 and 88) with $\mathbf{n}_\phi$ as the characteristic function. We also use a simple forward Euler discretization for the time derivatives with a time step size of $\Delta\tau = 0.5$. Because we use upwinding, information travels out from the level set, so if we wish to extend the velocity out to a width of N grid cells it will take $N/\Delta\tau$ time steps for the information to propagate out that far.

6.4 Surface Extraction

To extract triangles from the level set we use a simple exhaustive enumeration technique similar to marching cubes ([6] sec. 4.2). To avoid coding all 256 possible triangulations for cubes, and also to avoid dealing with the ambiguities from a naïve marching cubes implementation, we break each

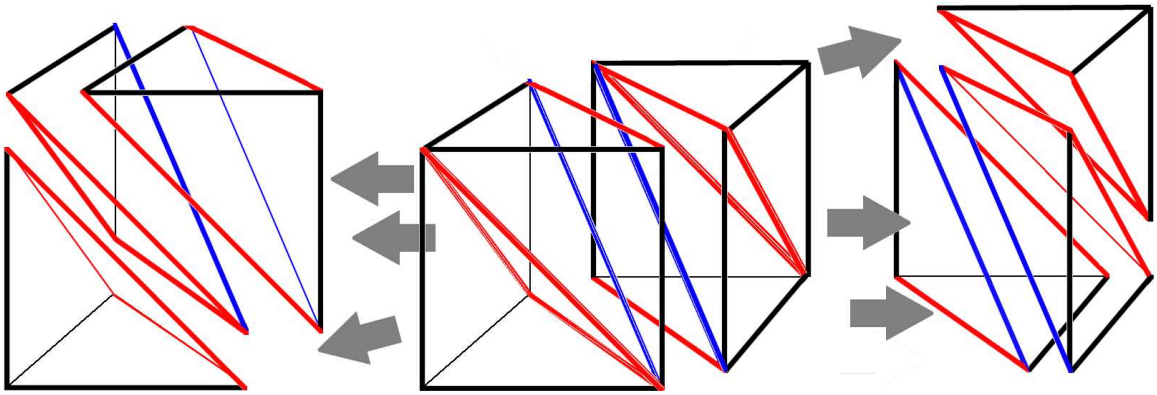

Figure 33: A cube is broken into 6 tetrahedra by adding diagonal edges to each face (red edges) and an edge connecting the opposite corners of the cube (blue edge).

cube into a number of tetrahedra. There are only 16 triangulations for an implicit surface passing through a tetrahedron, and with similarities between the cases there are only 3. Most of the time we break the cube into 6 tetrahedra just like in [6] section 4.3.5, and shown in figure 33. The top right corner of figure 34 shows a surface extracted from a splash that has a tear forming in it. There are grid artifacts in that triangulation. To get rid of the artifacts a higher resolution level set could be used, for example if the fluid computational grid is $24 \times 24 \times 24$ then the level set grid could be computed on a $48 \times 48 \times 48$ grid. However, we decided to use the same grid resolution for the level set and the computational grid, and instead we used the high frequency information that is given to us for free by the particles from the particle level set.

To get information from the particles we add vertices at the faces and center of each cube. First we get the interpolated level set values at those nodes then we use the standard error correction technique from the particle level set [14] to correct the level set value at the new vertices. The new vertices are then used to create a tetrahedral decomposition with 24 tetrahedra instead 6. There are 4 tetrahedra created for each face of the 6 cube faces as follows: all 24 tetrahedra share the vertex at the center of the cell, and each of the four tetrahedra that share a face use the vertex at the center of that face. The last two vertices for each of the tetrahedra come from the one of the four edges that make up the cube face. We tried a tetrahedral decomposition with 48 tetrahedra, by breaking each cube into 8 smaller cubes and then using the 6-tetrahedra configuration for each smaller cube, interpolating and correcting the level set on all added vertices. We also did the same thing with the 24-tetrahedra configuration creating a decomposition with a total of 192 tetrahedra. Figure 34 shows

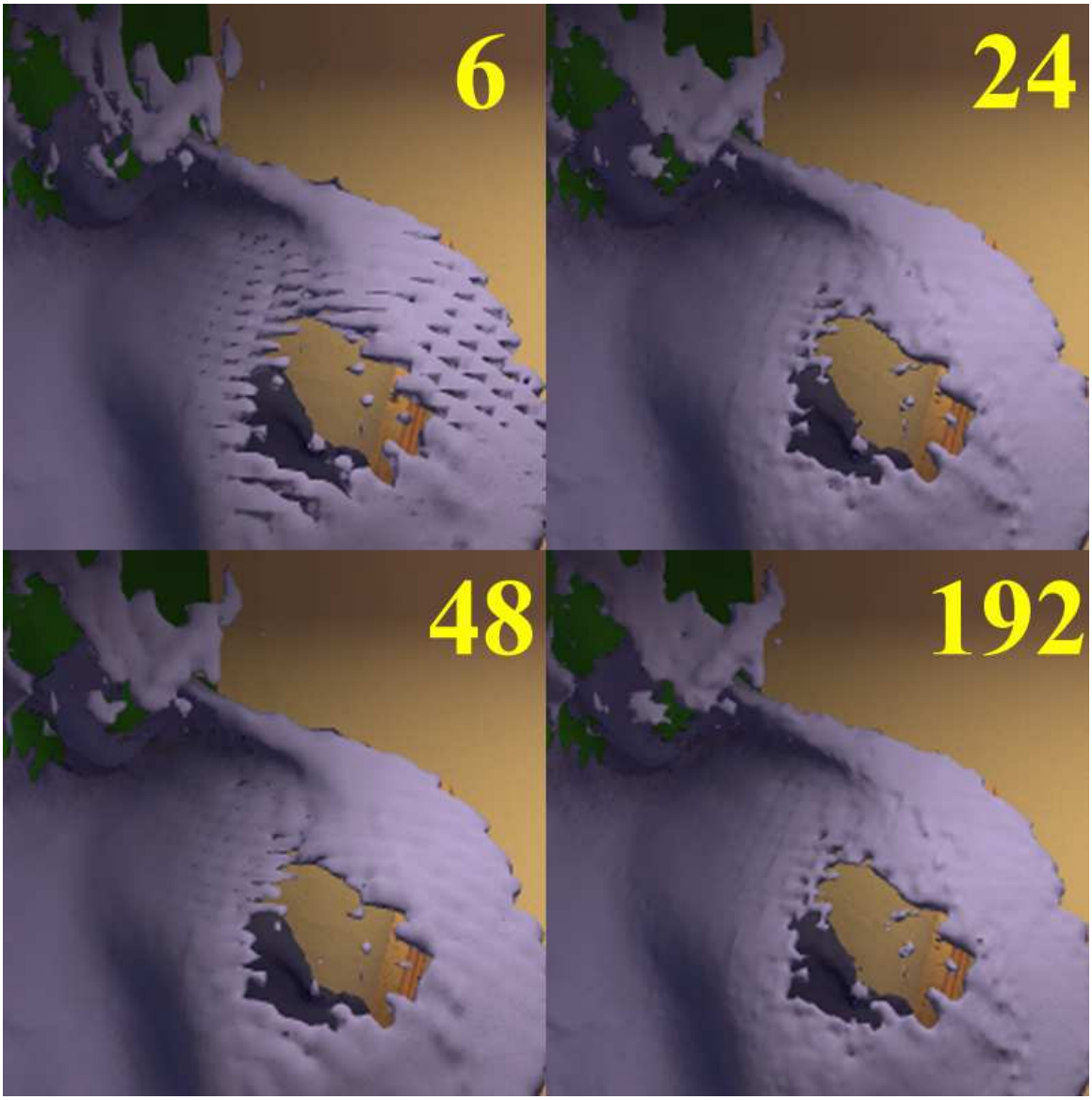

Figure 34: Four different surface extractions for the same splash as in figure 23. The water is rendered opaque so it is easier to see the detail, and the yellow numbers are the number of tetrahedra used per cube for each triangulation.

the same surface with all four configurations used. Although there is no noticeable improvement after the 24-tetrahedra configuration, we believe this is because we only used 32 particles per MAC grid cell in the particle level set technique, and that more particles (or careful reseeding of those particles) will allow for more information to be retrieved.

REFERENCES

[1] ADALSTEINSSON, D. and SETHIAN, J., "A fast level set method for propagating interfaces," *Journal of Computational Physics*, vol. 118, pp. 269–277, 1995.

[2] ARFKEN, G. B. and WEBER, H. J., *Mathematical Methods for Physicists*. San Diego, California: Academic Press, fourth ed., 1995.

[3] ARORA, M. and ROE, P. L., "A well-behaved tvd limiter for high-resolution calculations of unsteady flow," *Journal of Computational Physics*, vol. 132, pp. 3–11, 1997.

[4] BARRETT, R., BERRY, M., CHAN, T. F., DEMMEL, J., DONATO, J., DONGARRA, J., EIJKHOUT, V., POZO, R., ROMINE, C., and VAN DER VORST, H., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[5] BELDING, T. C., "Numerical replication of computer simulations: Some pitfalls and how to avoid them," Report PSCS-2000-001, Center for the Study of Complex Systems at the University of Michigan, 2000.

[6] BLOOMENTHAL, J. and WYVILL, B., *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

[7] CARLSON, M., MUCHA, P. J., and TURK, G., "Rigid fluid: Animating the interplay between rigid bodies and fluid," *ACM Transactions on Graphics*, vol. 23, pp. 377–384, Aug. 2004.

[8] CARLSON, M., MUCHA, P. J., VAN HORN III, R. B., and TURK, G., "Melting and flowing," in *ACM SIGGRAPH Symposium on Computer Animation*, pp. 167–174, July 2002.

[9] CHEN, J. X. and DA VITORIA LOBO, N., "Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations," *Graphical Models and Image Processing*, vol. 57, pp. 107–116, Mar. 1995.

[10] CHORIN, A. J., "Numerical solutions of Navier-Stokes equations," *Math. Comp.*, vol. 22, pp. 49–73, 1968.

[11] CHUNG, T. J., *Computational Fluid Dynamics*. Cambridge, United Kingdom: Cambridge University Press, 2002.

[12] COHEN, J. M. and MOLEMAKER, M. J., "Practical simulation of surface tension flows," in *SIGGRAPH 2004: Sketches*, 2004.

[13] DESBRUN, M. and GASCUEL, M.-P., "Animating soft substances with implicit surfaces," *Computer Graphics*, vol. 29, no. Annual Conference Series, pp. 287–290, 1995.

[14] ENRIGHT, D., FEDKIW, R., FERZIGER, J., and MITCHELL, I., "A hybrid particle level set method for improved interface capturing," *Journal of Computational Physics*, vol. 183, pp. 83–116, 2002.

[15] ENRIGHT, D., LOSASSO, F., and FEDKIW, R., "A fast and accurate semi-lagrangian particle level set method," *Computers and Structures (in press)*, 2004.

[16] ENRIGHT, D. P., MARSCHNER, S. R., and FEDKIW, R. P., "Animation and rendering of complex water surfaces," in *SIGGRAPH 2002 Conference Proceedings* (HUGHES, J., ed.), Annual Conference Series, pp. 736–744, ACM Press/ACM SIGGRAPH, 2002.

[17] ENRIGHT, D. P., *Use of the Particle Level Set Method For Enhanced Resolution of Free Surface Flows*. PhD thesis, Stanford University, Stanford, California, Aug. 2002.

[18] ERIKSSON, K., ESTEP, D., HANSBO, P., and JOHNSON, C., eds., *Computational Differential Equations*. Cambridge University Press, 1996.

[19] FATTAL, R. and LISCHINSKI, D., "Target-driven smoke animation," *ACM Transactions on Graphics*, vol. 23, pp. 441–448, Aug. 2004.

[20] FEDKIW, R., STAM, J., and JENSEN, H. W., "Visual simulation of smoke," in *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pp. 15–22, Aug. 2001.

[21] FEDKIW, R. P., "Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method," *Journal of Computational Physics*, vol. 175, pp. 200–224, 2002.

[22] FELDMAN, B. E., O'BRIEN, J. F., and ARIKAN, O., "Animating suspended particle explosions," in *Proceedings of ACM SIGGRAPH 2003*, pp. 708–715, Aug. 2003.

[23] FÄLT, H. and ROBLE, D., "Fluids with extreme viscosity," in *SIGGRAPH 2003: Sketches & Applications*, 2003.

[24] FOSTER, N. and FEDKIW, R., "Practical animation of liquids," in *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pp. 23–30, Aug. 2001.

[25] FOSTER, N. and METAXAS, D., "Realistic animation of liquids," *Graphical Models and Image Processing*, vol. 58, no. 5, pp. 471–483, 1996.

[26] FOSTER, N. and METAXAS, D., "Controlling fluid animation," in *Proceedings CGI '97*, pp. 178–188, 1997. Winner of the Androme Award 1997.

[27] FOSTER, N. and METAXAS, D., "Modeling the motion of a hot, turbulent gas," *Computer Graphics*, vol. 31, no. Annual Conference Series, pp. 181–188, 1997.

[28] FOURNIER, A. and REEVES, W. T., "A simple model of ocean waves," *Computer Graphics*, vol. 20, pp. 75–84, Aug. 1986.

[29] FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., and JONES, T. R., "Adaptively sampled distance fields: a general representation of shape for computer graphics," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 249–254, ACM Press/Addison-Wesley Publishing Co., 2000.

[30] GÉNEVAUX, O., HABIBI, A., and DISCHLER, J.-M., "Simulating fluid-solid interaction," in *Graphics Interface*, pp. 31–38, CIPS, Canadian Human-Computer Communication Society, A K Peters, June 2003.

[31] GLOWINSKI, R., PAN, T.-W., HESLA, T. I., and JOSEPH, D. D., "A distributed Lagrange multiplier/fictious domain method for particulate flows," *International Journal of Multiphase Flow*, vol. 25, pp. 755–794, Aug. 1999.

[32] GOKTEKIN, T., BARGTEIL, A. W., and O'BRIEN, J. F., "A method for animating viscoelastic fluids," *ACM Transactions on Graphics*, vol. 23, pp. 463–467, Aug. 2004.

[33] GOLUB, G. H. and VAN LOAN, C. F., *Matrix Computation*. John Hopkins Studies in the Mathematical Sciences, Baltimore, Maryland: Johns Hopkins University Press, third ed., 1996.

[34] GREENWOOD, S. and HOUSE, D., "Better with bubbles: Enhancing the visual realism of simulated fluid," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[35] GRIEBEL, M., DORNSEIFER, T., and NEUNHOEFFER, T., *Numerical Simulation in Fluid Dynamics, a Practical Introduction*. Philadelphia, PA: SIAM Press, 1998.

[36] GUENDELMAN, E., BRIDSON, R., and FEDKIW, R. P., "Nonconvex rigid bodies with stacking," *ACM Transactions on Graphics*, vol. 22, pp. 871–878, July 2003.

[37] HARLOW, F. H. and WELCH, J. E., "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *Physics of Fluids*, vol. 8, pp. 2182–2189, Dec. 1965.

[38] HINSINGER, D., NEYRET, F., and CANI, M.-P., "Interactive animation of ocean waves," in *ACM SIGGRAPH Symposium on Computer Animation*, pp. 161–166, July 2002.

[39] HIRT, C., AMSDEN, A., and COOK, J., "An arbitrary Lagrangian-Eulerian computing method for all flow speeds," *Journal of Computational Physics*, vol. 14, pp. 227–253, 1974.

[40] HONG, J.-M. and KIM, C.-H., "Animation of bubbles in liquid," *Computer Graphics Forum*, vol. 22, pp. 253–463, Sept. 2003.

[41] HOUSTON, B., BOND, C., and WIEBE, M., "A unified approach for modeling complex occlusions in fluid simulation," in *SIGGRAPH 2003: Sketches & Applications*, 2003.

[42] IHM, I., KANG, B., and CHA, D., "Animation of reactive gaseous fluids through chemical kinetics," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[43] JENSEN, H. W., MARSCHNER, S. R., LEVOY, M., and HANRAHAN, P., "A practical model for subsurface light transport," in *SIGGRAPH 2001 Conference Proceedings, August 12–17, 2001, Los Angeles, CA* (ACM, ed.), (New York, NY 10036, USA), pp. 511–518, ACM Press, 2001.

[44] KASS, M. and MILLER, G., "Rapid, stable fluid dynamics for computer graphics," *Computer Graphics*, vol. 24, pp. 49–57, Aug. 1990.

[45] KUNIMATSU, A., WATANABE, T., FUJII, H., SAITO, T., HIWADA, K., TAKAHASHI, T., and UEKI, H., "Fast simulation and rendering techniques for fluid objects," in *EG 2001 Proceedings* (CHALMERS, A. and RHYNE, T.-M., eds.), vol. 20(3) of *Computer Graphics Forum*, pp. 57–66, Blackwell Publishing, 2001.

[46] LAMBERT, J. D., *Numerical methods for ordinary differential systems: the initial value problem*. New York, NY, USA; London, UK; Sydney, Australia: John Wiley and Sons, Inc., 1991.

[47] LOSASSO, F., GIBOU, F., and FEDKIW, R., "Simulating water and smoke with an octree data structure," *ACM Transactions on Graphics*, vol. 23, pp. 457–462, Aug. 2004.

[48] MCNAMARA, A., TREUILLE, A., POPOVIC, Z., and STAM, J., "Fluid control using the adjoint method," *ACM Transactions on Graphics*, vol. 23, pp. 449–456, Aug. 2004.

[49] MIHALEF, V., METAXAS, D., and SUSSMAN, M., "Animation and control of breaking waves," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[50] MILLER, G. and PEARCE, A., "Globular dynamics: A connected particle system for animating viscous fluids," *Computers and Graphics*, vol. 13, no. 3, pp. 305–309, 1989.

[51] MIRTICH, B., "Fast and accurate computation of polyhedral mass properties," *Journal of Graphics Tools*, vol. 1, no. 2, 1996. ISSN 1086-7651.

[52] MORTON, K. W. and MAYERS, D. F., eds., *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 1994.

[53] MÜLLER, M., CHARYPAR, D., and GROSS, M., "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA-03)* (BREEN, D. and LIN, M., eds.), (Aire-la-Ville), pp. 154–159, Eurographics Association, July 26–27 2003.

[54] NIXON, D. and LOBB, R., "A fluid-based soft-object model," *IEEE Computer Graphics and Applications*, vol. 22, pp. 68–75, July/Aug. 2002.

[55] O'BRIEN, J. F. and HODGINS, J. K., "Dynamic simulation of splashing fluids," in *Computer Animation '95*, pp. 198–205, Apr. 1995.

[56] O'BRIEN, J. F., ZORDAN, V. B., and HODGINS, J. K., "Combining active and passive simulations for secondary motion," *IEEE Computer Graphics and Applications*, vol. 20, pp. 86–96, July/Aug. 2000.

[57] OSHER, S. and SETHIAN, J. A., "Fronts propagating with curvature dependent speed: Algorithms based in hamilton-jacobi formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[58] OSHER, S. and FEDKIW, R. P., *Level Set Methods and Dynamic Implicit Surfaces*. No. 153 in Applied Mathematical Sciences, New York: Springer-Verlag, 2003.

[59] PANTON, R. L., *Incompressible Flow*. Wiley-Interscience, 2nd ed., 1997.

[60] PATANKAR, N. A., "A formulation for fast computations of rigid particulate flows," *Center for Turbulence Research Annual Research Briefs 2001*, pp. 185–196, 2001.

[61] PATANKAR, N. A., SINGH, P., JOSEPH, D. D., GLOWINSKI, R., and PAN, T.-W., "A new formulation of the distributed Lagrange multiplier/fictitious domain method for particulate flows," *International Journal of Multiphase Flow*, vol. 26, pp. 1509–1524, Sept. 2000.

[62] PEACHEY, D. R., "Modeling waves and surf," *Computer Graphics*, vol. 20, pp. 65–74, Aug. 1986.

[63] PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H., and KANG, M., "A PDE-based fast local level set method," *Journal of Computational Physics*, vol. 155, pp. 410–438, 1999.

[64] PESKIN, C. S., "The immersed boundary method," *Acta Numerica*, vol. 11, pp. 479–517, 2002.

[65] PEYRET, R. and TAYLOR, T. D., *Computational Methods for Fluid Flow*. Springer Series in Computational Physics, New York: Springer-Verlag, 1983.

[66] PIGHIN, F., COHEN, J. M., and SHAH, M., "Modeling and editing flows using advected radial basis functions," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[67] PREMOE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., and WHITAKER, R. T., "Particle-based simulation of fluids," *Computer Graphics Forum*, vol. 22, pp. 401–411, Sept. 2003.

[68] PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., and VETTERLING, W. T., *Numerical Recipes in C: The Art of Scientific Computing*. University Press, Cambridge, 1993.

[69] RASMUSSEN, N., N. D. G. W. and FEDKIW, R., "Smoke simulation for large scale phenomena," *ACM Transactions on Graphics*, vol. 22, pp. 703–707, July 2003.

[70] RASMUSSEN, N., ENRIGHT, D., NGUYEN, D., MARINO, S., SUMNER, N., GEIGER, W., HOON, S., and FEDKIW, R., "Directable photorealistic liquids," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[71] REYNOLDS, C. W., "Flocks, herds, and schools: A distributed behavioral model," vol. 21, pp. 25–34, July 1987.

[72] SAAD, Y., *Iterative Methods for Sparse Linear Systems*. Philadelphia: SIAM, 2nd ed., 2003.

[73] SETHIAN, J., *Level Sets Methods and Fast Marching Methods*. Cambridge University Press, 2nd ed., 1999.

[74] SETHIAN, J., "A fast marching level set method for monotonically advancing fronts," *Porceedings of the National Academy of Sciences*, vol. 93, pp. 1591–1595, 1996.

[75] SHAH, M., COHEN, J. M., PATEL, S., LEE, P., and PIGHIN, F., "Extended galilean invariance for adaptive fluid simulation," in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, p. in press, Aug. 2004.

[76] SHI, L. and YU, Y., "Visual smoke simulation with adaptive octree refinement," Tech. Rep. UIUCDCS-R-2002-2311, University of Illinois at Urbana-Champaign, Dec. 2002.

[77] SHU, C.-W. and OSHER, S., "Efficient implementation of essentially nonoscillatory shock-capturing schemes," *Journal of Computational Physics*, vol. 77, pp. 439–471, 1988.

[78] SINGH, P., HESLA, T. I., and JOSEPH, D. D., "Distributed Lagrange multiplier method for particulate flows with collisions," *International Journal of Multiphase Flow*, vol. 29, pp. 495–509, Mar. 2003.

[79] STAM, J., "Stable fluids," in *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pp. 121–128, Aug. 1999.

[80] STAM, J., "Interacting with smoke and fire in real time," *Communications of the ACM*, vol. 43, no. 7, pp. 76–83, 2000.

[81] STAM, J., "A simple fluid solver based on the fft," *Journal of Graphics Tools*, vol. 6, no. 2, pp. 43–52, 2001.

[82] STAM, J., "Flows on surfaces of arbitrary topology," *ACM Transactions on Graphics*, vol. 22, pp. 724–731, July 2003.

[83] STAM, J., "Real-time fluid dynamics for games," in *Proceedings of the Game Developer Conference*, Mar. 2003.

[84] STORA, D., AGLIATI, P.-O., CANI, M.-P., NEYRET, F., and GASCUEL, J.-D., "Animating lava flows," in *Graphics Interface*, pp. 203–210, 1999.

[85] SUMNER, N., HOON, S., GEIGER, W., MARINO, S., RASMUSSEN, N., and FEDKIW, R., "Melting a terminatrix," in *SIGGRAPH 2003: Sketches & Applications*, 2003.

[86] SUSSMAN, M., SMEREKA, P., and OSHER, S., "A level set approach for computing solutions to incompressible two-phase flow," *Journal of Computational Physics*, vol. 114, pp. 146–159, 1994.

[87] TAKAHASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., and UEKI, H., "Realistic animation of fluid with splash and foam," *Computer Graphics Forum*, vol. 22, pp. 391–401, Sept. 2003.

[88] TAKAHASHI, T., HEIHACHI, U., and KUNIMATSU, A., "The simulation of fluid-rigid body interaction," in *SIGGRAPH 2002: Sketches & Applications*, p. 266, 2002.

[89] TERZOPOULOS, D., PLATT, J., and FLEISCHER, K., "Heating and melting deformable models," *The Journal of Visualization and Computer Animation*, vol. 2, pp. 68–73, Apr.–June 1991.

[90] TERZOPOULOS, D., PLATT, J., and FLEISCHER, K., "Heating and melting deformable models (from goop to glop)," in *Proceedings of Graphics Interface '89*, pp. 219–226, June 1989.

[91] TONNESEN, D., "Modelling liquids and solids using termal particles," in *Graphics Interface'91*, pp. 255–262, 1991.

[92] TREFETHEN, L. N., *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations*. unpublished text, 1996. available at http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/pdetext.html.

[93] TREFETHEN, L. N., *Spectral Methods in MATLAB*. Software, environments, tools, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[94] TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., and STAM, J., "Keyframe control of smoke simulations," *ACM Transactions on Graphics*, vol. 22, pp. 716–723, July 2003.

[95] WEI, X., LI, W., and KAUFMAN, A., "Melting and flowing of viscous volumes," in *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, p. 54, IEEE Computer Society, 2003.

[96] WEI, X., LI, W., MUELLER, K., and KAUFMAN, A. E., "The lattice-Boltzmann method for simulating gaseous phenomena," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, pp. 164–176, Mar./Apr. 2004.

[97] WEI, X., ZHAO, Y., FAN, Z., LI, W., YOAKUM-STOVER, S., and KAUFMAN, A., "Blowing in the wind," in *Eurographics/SIGGRAPH Symposium on Computer Animation* (BREEN, D. and LIN, M., eds.), pp. 075–085, Eurographics Association, 2003.

[98] WEIMER, H. and WARREN, J., "Subdivision schemes for fluid flow," in *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pp. 111–120, Aug. 1999.

[99] WELCH, J. E., HARLOW, F. H., SHANNON, J. P., and DALY, B. J., "THE MAC METHOD a computing technique for solving viscous, incompressible, transient fluid-flow problems involving free surfaces," Report LA-3425, Los Alamos Scientific Laboratory, 1965.

[100] WHITAKER, R. T., "A level-set approach to 3D reconstruction from range data," *Int. J. Computer Vision*, vol. 29, pp. 203–231, 1998. First sparse-field algorithm.

[101] WITTING, P., "Computational fluid dynamics in a traditional animation environment," in *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pp. 129–136, Aug. 1999.

[102] WOOTEN, W. L. and HODGINS, J. K., "Animation of human diving," *Computer Graphics Forum*, vol. 15, no. 1, pp. 3–14, 1996.

[103] WRENNINGE, M., "Fluid simulation for visual effects," Master's thesis, Linköpings Universitet, Linköping, Sweden, Apr. 2003.

[104] WRENNINGE, M. and ROBLE, D., "Fluid simulation interaction techniques," in *SIGGRAPH 2003: Sketches & Applications*, 2003.

[105] YNGVE, G. D., O'BRIEN, J. F., and HODGINS, J. K., "Animating explosions," in *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pp. 29–36, 2000.

[106] ZALESAK, S. T., "Fully multidimensional flux-corrected transport algorithms for fluids," *Journal of Computational Physics*, vol. 31, pp. 335–362, 1979.

VITA

Mark Thomas Carlson was born August 1st 1974, two minutes after his brother Paul. Together they were the first set of twins born at the Leesburg Regional Medical Center in Florida. Their father, E. Thomas Carlson, M.D., and mother, Gail Ann Carlson, ARNP, raised the twins and their four younger siblings (Leigh, Jane, Matthew, and Luke) on the shores of Lake Sylvan in Eustis Florida. After graduating Eustis High school in 1993, Mark attended the University of Central Florida in Orlando, where he graduated, in 1997, with a minor in creative writing and a bachelors in computer science. While in Orlando, Mark also studied at the Wah Lum Kung Fu Temple of Grandmaster Pui Chan. After a semester of post-back work in Literature at UCF, Mark moved to Boston with his future fiancé, Kristi. Mark spent the year in Boston programming communication and diagnostic software for a government contractor while saving money for graduate school. In 1999 Mark started his graduate work at Georgia Tech, where he worked with Jessica Hodgins in the Animation Lab. Greg Turk became Mark's advisor in 2001 and, together with Peter J. Mucha, guided Mark though his years of graduate study.