

Animation of Reactive Fluids

by

William Franklin Gates

B.Sc. (Computer Science and Psychology), University of California at Davis, 1991

M.Sc. (Computer Science), University of British Columbia, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

April 2002

© William Franklin Gates, 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date APRIL 25, 2002

Abstract

This thesis presents a general method for the integrated computer animation of the shape and shading of multiple reactive fluids in complex environments. This method advances the state of the art in fluid animation in three basic ways. First, it applies to a much larger class of fluid phenomena than previously addressed in computer graphics: both gases and liquids containing chemically reactive species. Second, it integrates a simple yet powerful procedural animation method for modelling both chemical and thermal reactions and their effects on the appearance and behaviour of simulated fluids. Third, and perhaps most significantly, it provides greater control: the desired flow can be specified at any location as well as the degree to which the simulation should be constrained to match it. We use the Navier-Stokes equations for incompressible flow as a general model of fluid motion and numerically solve these equations with finite differences on a fixed, uniform grid using techniques adapted from computational fluid dynamics for the specific requirements of computer animation. We illustrate the effectiveness of our method by applying it to a number of scenarios that would be difficult to animate using existing techniques.

Contents

| | |
|---|------------|
| Abstract | ii |
| Contents | iii |
| List of Tables | ix |
| List of Figures | x |
| Acknowledgements | xv |
| Dedication | xvi |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Problem Statement | 4 |
| 1.3 Methodology | 4 |
| 1.4 Relation to Previous Work | 6 |
| 1.5 Contributions | 6 |
| 1.6 Outline of Thesis | 7 |
| 2 Previous Work | 9 |
| 2.1 Fluid Animation | 9 |

| | | |
|----------|--|-----------|
| 2.1.1 | Wave Models | 9 |
| 2.1.2 | Particle Systems | 10 |
| 2.1.3 | Flow Field Models | 10 |
| 2.1.4 | Flow Simulation | 11 |
| 2.2 | Combustion | 15 |
| 2.3 | Weathering | 17 |
| 2.4 | Erosion | 19 |
| 2.5 | Reaction-Diffusion Models | 20 |
| 2.6 | Cellular Texture Generation | 21 |
| 2.7 | Summary | 22 |
| 3 | Framework | 24 |
| 3.1 | An Integrated Fluid Model | 24 |
| 3.2 | Reactive Elements | 27 |
| 3.3 | The Basic Algorithm | 29 |
| 3.4 | Summary | 30 |
| 4 | Dynamic Model | 31 |
| 4.1 | Introduction | 31 |
| 4.2 | Basic Principles of Fluid Flow | 32 |
| 4.2.1 | Fluid | 32 |
| 4.2.2 | Density | 33 |
| 4.2.3 | Viscosity | 34 |
| 4.2.4 | Eulerian and Lagrangian Descriptions of Fluid Motion | 35 |
| 4.2.5 | Compressible and Incompressible Flow | 36 |
| 4.3 | Dynamic Model | 37 |

| | | |
|----------|---|-----------|
| 4.3.1 | Assumptions | 37 |
| 4.3.2 | Conservation of Mass | 40 |
| 4.3.3 | Conservation of Momentum | 41 |
| 4.3.4 | Conservation of Energy | 42 |
| 4.3.5 | Chemical Transport and Reactions | 43 |
| 4.4 | Initial-Boundary Value Problem | 43 |
| 4.4.1 | Equations | 44 |
| 4.4.2 | Initial Conditions | 45 |
| 4.4.3 | Boundary Conditions | 45 |
| 5 | Numerical Simulation | 46 |
| 5.1 | Introduction | 46 |
| 5.2 | Finite Differences | 47 |
| 5.3 | Approximation of Convective Transport | 49 |
| 5.4 | Discretization of the Domain | 50 |
| 5.5 | Discretization of the Derivatives | 53 |
| 5.5.1 | Continuity Equation | 53 |
| 5.5.2 | Momentum Equation | 54 |
| 5.5.3 | Energy Equation | 54 |
| 5.5.4 | Chemical Transport and Reactions | 55 |
| 5.6 | Algorithm | 55 |
| 5.7 | Boundary Conditions | 57 |
| 5.7.1 | No-Slip Boundary Conditions | 58 |
| 5.7.2 | Free-Slip Boundary Conditions | 58 |
| 5.7.3 | Outflow Boundary Conditions | 58 |
| 5.7.4 | Periodic Boundary Conditions | 58 |

| | | |
|----------|---|-----------|
| 5.8 | Free Surface Conditions | 59 |
| 5.8.1 | One Gas Neighbor (Six Configurations) | 59 |
| 5.8.2 | Two Gas Neighbors (15 Configurations) | 59 |
| 5.8.3 | Three Gas Neighbors (20 Configurations) | 60 |
| 5.8.4 | Four Gas Neighbors (15 Configurations) | 60 |
| 5.8.5 | Five Gas Neighbors (Six Configurations) | 60 |
| 5.8.6 | Six Gas Neighbors (One Configuration) | 60 |
| 5.9 | The Pressure Poisson Equation | 61 |
| 5.9.1 | Linear System | 62 |
| 5.9.2 | Iterative Solution Methods | 62 |
| 5.9.3 | Error Tolerance | 62 |
| 5.10 | Interface Tracking | 63 |
| 5.11 | Stability Condition | 64 |
| 5.12 | Results | 64 |
| 5.12.1 | Rate of Convergence | 71 |
| 5.12.2 | Variation in Element Motion with Grid Resolution | 72 |
| 5.12.3 | Variation in the Number of Liquid Cells Over Time | 72 |
| 5.13 | Discussion | 73 |
| 6 | Controlling Flow Simulation | 89 |
| 6.1 | Introduction | 89 |
| 6.2 | The Influence Field | 90 |
| 6.3 | Streamtube Flow Primitives | 93 |
| 6.4 | Results | 97 |
| 6.4.1 | Basic Examples | 97 |
| 6.4.2 | Varying the Degree of Control (α_C) | 100 |

| | | |
|----------|--|------------|
| 6.4.3 | Composition of Streamtube Flow Primitives | 107 |
| 6.4.4 | Goal-Directed Fluid Flow | 107 |
| 6.5 | Discussion | 115 |
| 7 | Shape and Shading | 117 |
| 7.1 | Basic Principles of Image Synthesis | 117 |
| 7.1.1 | Surface Scattering | 118 |
| 7.1.2 | Light Transport in Participating Media | 118 |
| 7.1.3 | Radiance Equation | 120 |
| 7.2 | Liquid Surface Model | 120 |
| 7.2.1 | Shading Model | 121 |
| 7.3 | Participating Media | 122 |
| 7.3.1 | Volume Density Model | 123 |
| 7.3.2 | Phase Functions | 123 |
| 7.4 | Rendering | 126 |
| 7.4.1 | Photon Tracing Pass | 126 |
| 7.4.2 | Ray Tracing Pass | 129 |
| 7.5 | Results | 131 |
| 8 | Reactive Element Programs | 140 |
| 8.1 | Introduction | 140 |
| 8.2 | Control Surfaces and Volumes | 141 |
| 8.3 | A Programming Language for Reactive Elements | 142 |
| 8.3.1 | Data Types | 142 |
| 8.3.2 | Predefined Variables | 142 |
| 8.3.3 | Operators | 144 |

| | |
|---|------------|
| 8.3.4 Built-in Functions | 144 |
| 8.4 Compilation | 145 |
| 8.5 Results | 148 |
| 9 Conclusions | 154 |
| 9.1 Summary | 154 |
| 9.2 Analysis | 155 |
| 9.3 Future Work | 157 |
| Notation | 158 |
| Glossary | 161 |
| Bibliography | 164 |
| Appendix A Free Surface Conditions | 172 |
| Index | 184 |

List of Tables

| | | |
|-----|---|-----|
| 3.1 | Spatially varying parameters of our material model ($\mathbf{x} \in \Omega$). | 26 |
| 3.2 | Material constants. | 27 |
| 3.3 | Programmable reactive element properties. | 28 |
| 4.1 | Densities ρ of common fluids at 20°C and one atmosphere. | 33 |
| 4.2 | Approximate viscosities μ of common fluids at 20°C. | 34 |
| 4.3 | Incompressible and compressible gaseous flow regimes. | 36 |
| 4.4 | Physical parameters of our fluid model. | 37 |
| 7.1 | Photon maps. | 128 |
| 8.1 | Predefined variables. | 143 |
| 8.2 | Operators in order of increasing precedence. | 143 |
| 8.3 | Built-in functions. | 144 |

List of Figures

| | | |
|------|---|----|
| 3.1 | Digraph showing relationship between independent (A), interdependent (B), and dependent (C) parameters of our material model. . . | 25 |
| 3.2 | Main steps in one iteration of the simulation algorithm. | 29 |
| 5.1 | Boundary cells (darker) and computational cells (lighter). A two-dimensional grid is shown for clarity; our methodology uses a three-dimensional grid. | 51 |
| 5.2 | Velocity components are staggered on the centers of cell faces. All other properties are taken at cell centers. | 52 |
| 5.3 | Indexing for staggered values on a two-dimensional cell (i, j) . A two-dimensional cell is shown for clarity; our methodology uses three-dimensional cells. | 53 |
| 5.4 | Steps in one iteration of the simulation algorithm. | 56 |
| 5.5 | First set of frames from Scenario A (dam break). | 66 |
| 5.6 | Second set of frames from Scenario A (dam break). | 67 |
| 5.7 | First set of frames from Scenario B. | 68 |
| 5.8 | Second set of frames from Scenario B. | 69 |
| 5.9 | Frames from Scenario C with a $20 \times 20 \times 20$ grid resolution. | 74 |
| 5.10 | Frames from Scenario D with a $50 \times 50 \times 50$ grid resolution. | 75 |

| | | |
|------|--|----|
| 5.11 | Rate of convergence for the solution of the pressure Poisson equation for the first frame in each simulation run of Scenario C requiring nontrivial solution (frame 100 for grid resolutions $10 \times 10 \times 10$ and $20 \times 20 \times 20$, frame 107 for grid resolution $30 \times 30 \times 30$, and frame 108 for grid resolution $40 \times 40 \times 40$ | 76 |
| 5.12 | Convergence rates for the solution of the pressure Poisson equation for various frames of the simulation run of Scenario C with the $20 \times 20 \times 20$ grid resolution. | 77 |
| 5.13 | A detail of Figure 5.12. | 78 |
| 5.14 | Number of iterations required to solve the pressure Poisson equation for Scenario C at various grid resolutions ($\epsilon = 0.0001$). | 79 |
| 5.15 | The divergence in liquid cells at each frame for various grid resolutions of Scenario C. | 80 |
| 5.16 | Number of iterations required to solve the pressure Poisson equation for Scenario D at various grid resolutions ($\epsilon = 0.0001$). | 81 |
| 5.17 | Tracer elements used in Scenario C. | 82 |
| 5.18 | Mean difference in the positions of “tracer” elements with respect to their positions at each frame in the reference grid resolution ($40 \times 40 \times 40$) for Scenario C. | 83 |
| 5.19 | Mean difference in the positions of “tracer” elements with respect to their positions at each frame in the reference grid resolution ($50 \times 50 \times 50$) for Scenario D. | 84 |
| 5.20 | Ratio of the number of liquid cells to the total number of grid cells for each frame in Scenario C at various grid resolutions. | 85 |

| | | |
|------|--|-----|
| 5.21 | Ratio of the number of liquid cells to the total number of grid cells for each frame in Scenario D at various grid resolutions. | 86 |
| 6.1 | A simple streamtube flow primitive and its control points. | 93 |
| 6.2 | A more complex streamtube flow primitive | 94 |
| 6.3 | Flow in a streamtube primitive. | 94 |
| 6.4 | Flow in another streamtube primitive. | 95 |
| 6.5 | The streamtube primitive specifies the desired flow within the tube; the simulation must solve for the flow throughout the domain that matches the flow specified in the streamtube. | 97 |
| 6.6 | Elements following the desired flow specified by the streamtube prim- itives. Achieving this degree of control using forces would be quite difficult. | 98 |
| 6.7 | A streamtube primitive can cross itself; the simulation must still com- pute incompressible flow. | 99 |
| 6.8 | Streamtube flow primitive at time $t = 0$ | 101 |
| 6.9 | Streamtube flow primitive at time $t = 0.03$ | 102 |
| 6.10 | Streamtube flow primitive at time $t = 0.33$ | 103 |
| 6.11 | A streamtube flow primitive with $\alpha_C = 0.08$ surrounds a point heat source at the center of the domain (time $t = 0.9$). | 104 |
| 6.12 | A streamtube flow primitive with $\alpha_C = 0.08$ surrounds a point heat source at the center of the domain (time $t = 1.9$). | 105 |
| 6.13 | A streamtube flow primitive with $\alpha_C = 0.08$ surrounds a point heat source at the center of the domain (time $t = 8.57$). | 106 |
| 6.14 | A streamtube flow primitive with $\alpha_C = 0.43$ surrounds a point heat source at the center of the domain (time $t = 2.13$). | 108 |

| | | |
|------|--|-----|
| 6.15 | A streamtube flow primitive with $\alpha_C = 0.43$ surrounds a point heat source at the center of the domain (time $t = 8.4$). | 109 |
| 6.16 | A streamtube flow primitive with $\alpha_C = 1$ at its boundary and $\alpha_C = 0$ at its center surrounds a point heat source at the center of the domain (time $t = 8.26$). | 110 |
| 6.17 | Composition of two streamtubes with constant $\alpha_C = 0.28$ | 111 |
| 6.18 | Composition of two streamtubes with radially decaying α_C , from $\alpha_C = 0.5$ to $\alpha_C = 0.0$, at time $t = 0.0$ | 112 |
| 6.19 | Composition of two streamtubes with radially decaying α_C , from $\alpha_C = 0.5$ to $\alpha_C = 0.0$, at time $t = 14.13$ | 113 |
| 6.20 | Two streamtube flow primitives with the same flow magnitude that have opposing flow directions in the region where they overlap. Both primitives have $\alpha_C = 0.3$ | 114 |
| 6.21 | An example of goal-directed flow simulation. | 116 |
| 7.1 | First set of frames from liquid drop sequence. | 133 |
| 7.2 | Second set of frames from liquid drop sequence. | 134 |
| 7.3 | Frames from sequence. | 135 |
| 7.4 | First set of frames from gas sequence. | 136 |
| 7.5 | Second set of frames from gas sequence. | 137 |
| 7.6 | Another gas sequence. | 138 |
| 7.7 | Liquid and gas simulation. | 139 |
| 8.1 | Reaction A source. | 146 |
| 8.2 | Reaction A compiled. | 146 |
| 8.3 | Reaction B source. | 147 |

| | | |
|-----|--|-----|
| 8.4 | Reaction B compiled. | 147 |
| 8.5 | First set of frames from Reaction A sequence. | 150 |
| 8.6 | Second set of frames from Reaction A sequence. | 151 |
| 8.7 | First set of frames from Reaction B sequence. | 152 |
| 8.8 | Second set of frames from Reaction B sequence. | 153 |

Acknowledgements

Alain Fournier, my supervisor, inspired me with his passion for science and humbled me with his kind spirit. His door was always open, and I remember with deep fondness our long conversations filled with ideas and humour and enlightening detours. He was a friend, and I miss him.

Taking on the role of acting supervisor, Kellogg S. Booth, went way beyond the call of duty, pushing me on, providing support and guidance, and seeing me through to the end. I could not have made it without him. The rest of my supervisory committee, James Varah, David Forsey, and Rosemary Knight, as well as my university examiners, Dinesh Pai and Ian Gartshore, provided valuable input that improved this dissertation. My external examiner, Nick Foster, was extraordinarily helpful: he flew to Vancouver to attend my defense and made many key suggestions.

My friends have been supportive, and each one has been a pleasure to know: Robert Scharein, Robert Walker, Gwen Litchfield, David Bullock, Chris Romanzin, Roger Tam, Lisa Streit, Chris Healey, Raza Khan, Scott Ralph, Valerie Summers, Lori Peterson, Tara Law, Vishwa Ranjan, Kevin Arthur, Amer Hussain, Jerry Bertaina, Brian Harrington, and Ed Stone. I am also glad to have known Peter Cahoon—colleague, poet, and neighbor—who passed away on January 1, 2000.

My family has been wonderful as always. Mom, Dad, Bob, Cindi, Fred, Devyn, and Davis: I love you all.

Finally, for financial support, I wish to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Department of Computer Science at the University of British Columbia.

WILLIAM FRANKLIN GATES

The University of British Columbia
April 2002



In memory of Alain Fournier (1943-2000)

Chapter 1

Introduction

1.1 Motivation

The realistic computer animation of fluids (liquids and gases) presents some of the most difficult challenges in computer graphics. The sheer complexity of fluid flow can make the use of standard animation methods painstaking. Imagine, for example, interactively positioning thousands of control points of a model of a breaking wave at every key frame. Such macroscopic complexity is an intrinsic characteristic of fluid: the molecules of a fluid have greater kinetic energy than those of a solid, allowing them to overcome attractive intermolecular forces and move freely past each other. Thus, fluids have an effectively infinite number of degrees of freedom. Fluid flow, however, is not random but has a characteristic appearance that is easy to recognize. Thus, to animate fluids by explicitly specifying how the geometry (shape) and appearance (shading) changes over time requires great skill and patience so that the resulting animations appears realistic.

The difficulty of animating complex flows in a convincing manner using standard methods has motivated dedicated methods for fluids which introduce specific

tools for animating the shape and shading of gases and liquids. Early work in fluid animation focused on non-dynamic, ad hoc models of water waves and flow fields as well as simple dynamic simulation methods based on systems of point masses. These techniques address a limited range of fluid motion. Recent research has clearly demonstrated the effectiveness of adapting techniques from computational fluid dynamics for the realistic animation of the general motion of liquids and gases [19, 22, 23, 26, 68, 89]. These techniques numerically solve the Navier-Stokes equations over time on a fixed, regular three-dimensional grid given initial and boundary conditions.

While this recent research has rapidly advanced the state of the art, fluid animation remains one of the most difficult tasks in computer graphics. This dissertation focuses on three fundamental challenges of animating fluids that have received little attention:

1. Integrated Animation of Fluids

Previous research in computer graphics has done little to address the integrated animation of the shape and shading of liquids and gases. Recent research has focused exclusively on the animation of a single fluid, most often specifically addressing either a gas or a liquid but not both. It is not clear how these techniques, even the most recent work on animating smoke [19] and liquid [22], can be integrated or extended to animate multiple fluids with different material properties. In addition, the shading of fluids has largely been considered an independent task, despite being dependent on many of the same physical properties that affect fluid shape.

2. Animation of Reactive Flow

Chemical and thermal reactions in fluids are responsible for a spectacular diversity of effects on the appearance and behaviour of gases and liquids. Combustion is perhaps the most dramatic example of reactive flow. Some specific effects have been addressed by ad hoc methods, e.g., models of flame [40, 59, 60, 62] and the propagation of fire [9, 61, 70]. These methods are difficult to generalize. Yngve et al. [89] modelled explosions based on the simulation of compressible flow. Weathering models have simulated the chemical and mechanical actions of liquids on solid surfaces [14, 15]. No research in computer graphics (to the author's knowledge), however, has involved the simulation of Navier-Stokes equations in conjunction with chemical reactions. An additional challenge is coupling the effects of changing chemical concentrations with the shading of fluids.

3. Control of Flow Simulation

The essence of animation is control. Scripted scenes often call for very specific flow behaviours such as a body of water flowing into the shape of a human face. This degree of control is effectively impossible to achieve by simply setting the initial and boundary conditions of a flow simulation. Foster and Metaxas [25] demonstrated a number of approaches to controlling their flow simulation including modelling body forces and adjusting pressure values. However, the effects of body forces and pressure modifications are often hard to predict. Foster and Fedkiw [22] showed how "fake" objects could be employed in their liquid simulation to direct the flow, but they noted that their approach "[does not give] perfect direct control over the liquid motion." The difficulty of controlling flow simulation is perhaps the most significant limitation of current techniques for animating fluids: animators are often reluctant to give up ex-

plicit control and are unwilling to fight unruly simulation methods that give unpredictable results.

1.2 Problem Statement

This dissertation addresses the integrated animation of the shape and shading of reactive gases and liquids in complex environments. The goal here is not to model real fluids but *to facilitate the realistic modelling of the appearance of fluids flowing in a desired manner*. To satisfy this goal we seek a coherent animation methodology based on the numerical simulation of the Navier-Stokes equations in three dimensions as well as an effective means of controlling the simulation. We require appropriate approximations to the full Navier-Stokes equations and their numerical solution for the specific requirements of animating the shape and shading of fluids.

1.3 Methodology

Our methodology is based on a general fluid model specifically designed for computer animation. This model integrates physical fluid properties such as velocity, pressure, and temperature with shape and shading attributes as well as abstract parameters for simulation control. All parameters of our model are defined over a fixed, three-dimensional spatial domain as scalar or vector fields.

Physical fluid properties are governed by the Navier-Stokes equations. We numerically solve these equations using a classic projection scheme [10] where at every time step we first compute an intermediate velocity field without regard for mass conservation and then project out a mass-conserving velocity field. We discretely approximate the continuous partial differential equations using finite differences on

a fixed, uniform grid. We use volume primitives we call *reactive elements* to track the evolution of material properties in the flow. Each element is associated with a specific material.

Reactive elements also serve other key roles in our methodology. They provide a simple but effective solution to a crucial problem: the interactive visualization of unsteady flow. They are also integral to our shape and shading models. And finally, they can be *programmed*, i.e., element properties can be animated with associated procedures. These procedures are programs in a dedicated high level language that are executed at every time step in the simulation. They provide a general mechanism of directly modelling the effects of chemical reactions (or any other real or imagined phenomenon) on the appearance and behaviour of flowing fluids.

Programming reactive elements is one of the two mechanisms that we introduce for controlling flow simulation. The other involves coupling user-modelled flow fields with our simulation in order to constrain it towards desired results. This approach corresponds to modelling implied external body forces, but rather than modelling forces whose effects may be hard to predict, we allow the direct specification of the desired influence on the fluid velocity using “flow primitives”.

Our animation method evolves our fluid model over time given initial and boundary conditions and a desired time step. The simulation iteratively advances through time, constrained by a standard stability condition that may require sub-step iterations. At the end of every time step, we render images using a volume ray-tracing algorithm.

1.4 Relation to Previous Work

This dissertation builds on ideas and techniques from previous work in fluid animation and other areas of computer graphics. The basic framework of our methodology is inspired in part by the work of Fleischer et al. [21] on modelling surface elements such as thorns, scales, and feathers with “cellular particles” governed by “cell programs” in an “extracellular environment” (this is discussed in Section 2.6). We have found the basic abstraction of programmable elements in a global environment to be quite useful. Our fluid flow simulation method is similar to recent work in fluid animation [19, 22, 68]: we use the same basic projection method and semi-Lagrangian integration scheme adopted from computational fluid dynamics as we feel that these techniques are good choices for computer animation. However, our simulation method addresses a significantly larger class of fluid phenomena than these earlier approaches.

1.5 Contributions

This dissertation makes three main contributions that advance the state of the art in the animation of fluids:

- We present a general method for the *integrated* animation of a much larger class of fluid phenomena than previously addressed in computer graphics: gaseous flow, free-surface liquid flow, and chemical and thermal reactions in both gases and liquids. Moreover, we address both the shape and shading of fluids.
- We present an open-ended approach to the modelling of chemical and thermal reactions and their effects on the appearance and behaviour of our simulated

fluid flows through the programming of reactive elements. This procedural animation method does not require any knowledge of fluid mechanics or chemical kinematics. Very simple procedures can give quite interesting results, and a diverse range of effects are possible.

- Perhaps the most significant contribution is an intuitive and powerful approach to controlling incompressible flow simulation. In this approach the desired fluid velocity can be specified at any location as well as the degree to which the simulation should be constrained to match it, allowing a continuum of simulation control from explicitly specifying the flow to specifying a subtle influence on it that generally directs its behaviour.

1.6 Outline of Thesis

After reviewing related research in Chapter 2, we present the conceptual framework of our animation methodology in Chapter 3. This framework defines an integrated fluid model consisting of physical properties, shape and shading attributes, and control parameters. The physically based component of our model is described in Chapter 4, which reviews the basic principles and equations of fluid dynamics and formulates our dynamic model as an initial-boundary value problem. The numerical solution of this problem using techniques from computational fluid dynamics is discussed in Chapter 5. As nonlinear flow simulation can be difficult to control for animation purposes, we present a technique for “directing” the flow to “perform” as desired in Chapter 6. The shape and shading component of our fluid model is described in Chapter 7, which also discusses our implementation of a bidirectional ray-tracing method for rendering realistic images from our fluid model. We present

a method of modelling chemical reactions and their effects on the flow using simple procedures in a dedicated programming language in Chapter 8. We discuss our results and suggest possible directions for future work in Chapter 9.

Chapter 2

Previous Work

In this chapter, we review previous work on animating fluids and other research in computer graphics related to this dissertation.

2.1 Fluid Animation

Recent research has demonstrated the effectiveness of animating gases and liquids using techniques adapted from computational fluid dynamics for numerically solving the Navier-Stokes equations in three dimensions. Our review focuses on this research as we use similar techniques, but first we briefly survey other approaches to animating fluids.

2.1.1 Wave Models

Early work modelled waves using parametric surfaces animated by ad hoc functions [49, 29, 58, 77]. For example, Fournier and Reeves [29] modelled ocean waves approaching a shore using functions that accounted for wave refraction with depth and the effects of wind on wave crests. These ad hoc models can be computationally

efficient but address a limited range of motion.

2.1.2 Particle Systems

Reeves [62] introduced the idea of using particle systems to model fire and other “fuzzy objects” where particles are point masses with associated properties and their motion is governed by dynamic simulation. Such particle systems are a simple and popular method of animating gases and liquids as well as myriad other phenomena such as fireworks and swarms of bees. A liquid surface is typically represented using an implicit surface where the implicit field function is generated by summing radially decaying field functions associated with each particle. The resulting surface usually has a characteristic “blobby” look.

Later work introduced intermolecular forces (attractive at long ranges and repulsive at short ranges) between particles to simulate viscous flow [51, 75]. This work was extended to include thermal effects where increasing temperature decreases the attraction between particles [76] and to account for volume conservation in the implicit surface generated from the particles [13]. For any practical number of particles, it is not clear that molecular dynamics is an adequate model of macroscopic behaviour. An alternate approach is to use continuum dynamics and a Lagrangian simulation method, e.g., lava flows have been animated using smoothed particle hydrodynamics (SPH), a Lagrangian fluid flow simulation method typically used for astrophysical problems [73].

2.1.3 Flow Field Models

A flow field describes fluid velocity as a function of space. A variety of non-dynamic methods of modelling flow fields have been developed for animating fluids and ob-

jects in fluids. Non-physical flow fields have been modelled using ad hoc techniques [16, 17, 18, 39, 67]. Mass-conserving flows fields for incompressible flows have been modelled by the superposition of “flow primitives” such as sources, sinks, and vortices [33, 84] and subdivision schemes that start with initial “sketches” of the flow [83]. These methods can be effective, but they do not model the dynamics of the flow: the burden is on the animator.

Turbulent flow fields have been modelled using stochastic methods [64, 66, 69, 70]. A typical approach is *Fourier synthesis* where the spectral characteristics of the motion are modelled in Fourier space and then a Fourier transformation is performed to generate a flow field.

2.1.4 Flow Simulation

The most effective approach to the computer animation of general fluid flow has been to adapt techniques from computational fluid dynamics (CFD) for the needs of computer graphics where the goal is to realistically simulate the *appearance* of flowing fluids. Only in recent years with advances in computing power has the fully three-dimensional simulation of the Navier-Stokes equations become practical for computer animation. Still many CFD techniques, especially for the dramatic flows often of interest in computer graphics (e.g., turbulent, free-surface, reactive flows), remain impractical for animation purposes where simulation is used as a tool for generating desired motion—generally a trial-and-error process requiring many simulation runs.

Earlier work in computer graphics addressed purely two-dimensional fluid flow simulation [32, 88] and liquid simulation where the liquid surface is constrained to be a height field function $z = f(x, y)$ where z is the height of the liquid surface

with respect to a point (x, y) in the plane. Kass and Miller [44] animated such height field models by simplifying hydrodynamic equations to the two-dimensional wave equation, discretely approximating it using finite differences, and then using a stable implicit numerical integration scheme to time-evolve the system. O'Brien and Hodgins [56] presented a similar approach. An alternate approach to animating a height field model of liquid is to numerically solve the Navier-Stokes equations in two dimensions and use scaled pressure values as height field elevations [7, 8], but this approach does not accurately model three-dimensional liquid motion. Height field models can be effective and computationally efficient for applicable scenarios, but obviously they can only represent a limited range of liquid motion.

Foster and Metaxas (1996)

Foster and Metaxas [23, 24] were the first to show that numerical simulation of the Navier-Stokes equations in three dimensions was a practical method of animating free-surface liquid flow. They used a classic method from computational fluid dynamics known as the marker-and-cell (MAC) method, originally developed in 1965 by Harlow and Welch [37]. In this method the Navier-Stokes equations are discretely approximated using finite differences over a fixed grid. The simulation algorithm is explicit in time and is stable subject to a Courant condition. The computed fluid velocity is used to govern massless marker particles or a height field to track the free liquid surface. In subsequent work, they presented several methods of controlling the liquid flow simulation using boundary conditions, external forces, and pressure modifications [25].

Foster and Metaxas (1997)

Foster and Metaxas [26] presented a similar method for animating gases that accounts for drag (shearing forces) caused by a faster moving gas mixing with a slower moving gas, thermal buoyancy (hotter parts of a gas rising faster than colder parts), and turbulent effects caused by interaction with solid objects. The grid resolution is a limiting factor in modelling effects such as rotational motion.

Stam (1999)

Stam [68] animated gaseous flow with an *unconditionally stable* method of numerically solving the incompressible Navier-Stokes equations. This method allows much larger time steps to be taken than would be possible with the method of Foster and Metaxas [26]. To achieve this stability, a semi-Lagrangian integration scheme is used to treat the convective terms in the equations and implicit methods are used to handle the other terms. The solution algorithm is based on a standard projection method originally due to Chorin [10]. The semi-Lagrangian integration scheme introduces excessive numerical dissipation where the flow is dampened more rapidly than real flows, but Stam [68] argues this is an acceptable trade-off for stability. Internal boundaries and free-surfaces were not addressed in this work.

Witting (1999)

Witting [86] presented a production system for the computer animation of fluids in a traditional animation environment. This system is based on the two-dimensional simulation of compressible Navier-Stokes equations. Images are used to define initial velocity and temperature fields, and the simulated flows are used to transform images.

Yngve et al. (2000)

An explosion is a burst of energy released in the air that generates a blast wave front (a nearly discontinuous increase in pressure, temperature, and density) that propagates at supersonic speeds. Some work in computer graphics has addressed explosions without simulating gaseous flow [50, 54]. However, Yngve et al. [89] demonstrated that a more general approach is to simulate the propagation of blast waves in gases. They used a finite differences approximation of the Navier-Stokes equations for compressible flow over a fixed, three-dimensional grid and a standard explicit algorithm for numerically integrating the compressible equations. They note that special care must be taken to deal with the rapid pressure changes caused by the supersonic blast wave: Stam's method [68] is not appropriate because they need to track the sharp wave front profile; instead they use a donor-acceptor method for the convective terms. The initial conditions of their simulation are set by using a polygonal mesh to define the blast wave; the mesh is voxelized and used to define the region of high pressure or temperature. The effects of the blast wave on objects are addressed by computing the forces on object boundaries due to fluid pressure. Rigid body simulation as well as fracture simulation can then be used to generate the response of the object. The effects of solids on the fluid are handled by voxelizing triangular mesh models and displacing fluid as required. Yngve et al. also address various secondary effects such as the refraction of light caused by blast waves.

Fedkiw, Stam, and Jensen (2001)

Fedkiw, Stam, and Jensen [19] animated smoke by simulating inviscid, incompressible flow on a fixed, three-dimensional grid. They note that the effects of viscosity are negligible in gases, especially on coarse grids where numerical dissipation dom-

inates physical viscosity and molecular diffusion. Their method adopts the semi-Lagrangian method used by Stam [68] and is also based on a projection method [10]. Their main contribution is a vorticity confinement scheme that increases vorticity which otherwise gets damped out due to the excessive numerical dissipation. This scheme is based on a technique from the computational fluid dynamics literature for modelling turbulent flow around helicopters [72].

Foster and Fedkiw (2001)

Foster and Fedkiw [22] presented a general method for animating incompressible, viscous liquid flow with free surfaces in environments with complex boundaries and moving objects. They numerically solve the Navier-Stokes equations using finite differences, a fixed grid, and a projection method [10]. The key innovation of their work is their integrated use of a level set method and particles with no inertia to track the free liquid surface. They also address moving objects in the flow. They show how this method for moving objects can be adapted to control the flow.

2.2 Combustion

Fire is the manifestation of combustion in heat, flame, and light. Combustion (burning) is a rapid oxidation reaction. Earlier work in computer graphics developed ad hoc models of flame [40, 59, 60, 62]. More recent research has addressed the underlying process of combustion.

Chiba et al. (1994)

Chiba et al. [9] animated fire in two-dimensions using a two-dimensional array of “fuel” cells. Each cell models the amount of fuel it contains, its temperature, and the

binary state of “burning” or “not burning”. A fuel cell is “ignited” if it contains fuel and is above a threshold temperature. Once “burning”, a fuel cell creates “flame” particles and reduces its fuel. Fuel cells absorb heat from particles and transfer heat to neighboring cells. Flames and smoke are modelled using particle systems [62]. The forces on a particle include the effects of a “vortex field”, an “ascending current”, other particles, and damping. The vortex field models turbulent flow using a random distribution of vortex primitives that grow and decay in strength over their life cycles. The ascending current force models thermal influences on its motion; the force is proportional to temperature of the particle (which is affected by the temperature of nearby particles). Forces due to other particles model diffusion of particles with relatively low temperatures and convergence of particles with relatively high temperatures. Flame particles that cool below a threshold temperature are converted into smoke particles. At each time step, the particles are rendered using filtered line segments based on their paths during the time step.

Perry and Picard (1994)

Perry and Picard [61] presented coupled flame and fire spread models. As with Chiba et al. [9], flames are modelled using an approach based on particle systems [62]. The spread of fire is modelled by tracking the boundary between burning and non-burning regions. This boundary is represented by a ring of linked “control points” that are constrained both to stay on the surface of a polygonal model and to avoid any self-intersections in the boundary. The velocity of these control points depends on fuel density, temperature, the thermal conductivity of the gas, and other factors. Flame particles are generated from advancing control points.

Stam and Fiume (1995)

Stam and Fiume [70] presented a simple model of flame and fire spread. In their approach, they simulate the advection and diffusion of density fields for fuel, flame, and smoke in flow fields modelled by techniques discussed in Section 2.1.3. At a sufficiently high temperature, fuel begins to burn, converting fuel to flame. Flame evolves and cools, transferring heat to fuel, and when sufficiently cold, generates smoke. The generation of flame and smoke is based on *Arrhenius formula*. A “shooting operation” is performed to calculate the source term for fuel temperature. While their results are impressive, Stam and Fiume [70] note: “the fire model has a large set of interdependent parameters which are not necessarily easy to manipulate ... getting the fire look ‘just right’ can be problematic.”

2.3 Weathering

A common complaint about computer generated imagery is that the depicted scenes often appear too pristine. Researchers in computer graphics have developed various methods of modelling the effects of weathering. Here we review techniques that address fluid flow and chemical or mechanical fluid-solid interactions.

Washes and Stains Caused By Rain

Dorsey, Pedersen, and Hanrahan [15] presented a method for modelling surface materials that have been weathered by rain. They noted that the standard approach to modelling a weathered appearance, texture-mapping, suffers from several drawbacks:

... it is very difficult to match textures across surface boundaries or

account properly for area distortions caused by the parameterization.

More fundamentally, the texture mapping approach does not take into account the structure of properties of a given material or the processes in the surrounding environment that account for weathering.

Thus, they model water flowing over a surface model, dissolving and absorbing material, carrying it, and then depositing it back on the surface. This process creates weathering effects such as patterns of washes and stains. Both “base material” and “loose deposits” are modelled over a surface as supplemental texture maps. Water is modelled as particles that flow over the surface, affected by the roughness and absorbtivity of the base material. Particle positions are governed by dynamic simulation where forces include gravity, friction, self-repulsion, and diffusion. Force vectors are constrained to lie on the surface (by projecting them onto the tangent plane of the surface) except when particles fall off a surface to a lower one. When the mass of a particle falls below some threshold, it is removed from the simulation. Differential equations define the rates of absorption and sedimentation.

Weathering of Stone

The weathered appearance of stone is the result of a variety of chemical and mechanical effects. Rainwater dissolves oxides of carbon, sulphur, and nitrogen, forming a solution that is absorbed by the porous stone. This solution transforms the minerals in the stone into gypsum and other substances that are readily soluble in water. The recrystallization of these substances, combined with pollution, forms a crust on the stone surface which erodes over time.

Dorsey et al. [14] presented a method of modelling and rendering weathered stone that simulates these weathering processes. Their method is based on a mesh

of volume elements, “slabs”, that are created by extruding a quadrilateral mesh. An input surface model is sampled at positions with slabs, giving density values that define position outside, on, and inside the surface. Material properties are then generated at these sample positions using procedural techniques. The simulation of weathering process—moisture travel, dissolution and recrystallization of minerals, and erosion—is done using finite differences over the voxels in the slabs. The travel of moisture (through porous stone) is done by tracking a front between wet and dry regions. The stone model is rendered taking into account subsurface scattering using a Monte Carlo based ray marcher.

2.4 Erosion

Terrain modelling has received considerable interest in computer graphics. Early work modelled terrain using stochastic techniques, e.g., the work of Fournier et al. [28]. However, more recent research has used the simulation of erosion as a mechanism of generating terrain models. We review these erosion-based methods of terrain modelling with a focus on the modelling of liquid-solid interactions.

Musgrave, Kolb, and Mace [52] modelled terrain by generating an initial shape using fractal techniques and then using a simple erosion model to simulate the development of streams, valleys, and talus slopes. Their erosion model consists of two components: a hydraulic erosion model that simulates rain falling on the terrain and carrying sediment downhill, and a thermal weathering model that simulates other processes that cause material to dislodge and gather at the bottom of an incline.

Roudier, Peroche, and Perrin [63] presented a similar method of modelling terrain by simulating erosion. The shape of the terrain is modelled by a two-

dimensional hexagonal grid of elevation values (a height field model). The properties of the terrain are modelled by three-dimensional fields. Each grid node uses its associated elevation value to map into this “solid texture” of terrain properties. To model these properties, they assume the terrain consists of layered rocks (strata) deformed by folds and faults (continuous and discontinuous cylindrical deformations respectively). They use an hoc algorithm for simulating erosion and deposition. Nagashima [53] presented a similar terrain modelling method that requires fewer geological parameters to be specified.

2.5 Reaction-Diffusion Models

Little work in computer graphics has addressed the modelling of chemical reactions. A notable exception is pattern formation methods based on a reaction-diffusion model originally proposed in 1952 by Turing [78] for the development of biological patterns. In this model concentrations of hypothetical chemicals (“morphogens”) that control pigment production diffuse and react to form patterns such as spots and stripes. A basic formulation of this reaction-diffusion model for two morphogens with concentrations a and b is:

$$\begin{aligned}\frac{\partial a}{\partial t} &= f(a, b) + d_a \nabla^2 a \\ \frac{\partial b}{\partial t} &= g(a, b) + d_b \nabla^2 b\end{aligned}$$

where f and g are functions modelling the interaction of the two chemicals and d_a and d_b are the coefficients of diffusion.

Researchers in computer graphics have adopted this reaction-diffusion model for texture synthesis. Turk [79] showed how reaction-diffusion simulations could be used in a cascade fashion where the system generates the initial conditions of

other systems that refine the pattern. He also presented a method of matching the generated texture to the geometry of a polyhedral surface by running the simulation directly on the surface. Witkin and Kass [85] introduced anisotropic and spatially non-uniform diffusion and a method for mapping the texture to a parametric surface that accounts for non-uniform parameterization. Fowler, Meinhardt, and Prusinkiewicz [30] integrated a reaction-diffusion model into their modelling methods for sea shells. Reaction-diffusion models have been shown to be capable of generating many interesting patterns; however, generating a particular pattern can be difficult.

An alternative approach to synthesizing biological patterns, especially mammalian coat patterns, is the biologically motivated *clonal mosaic model* [80, 81]. In this approach, discrete cells are used to generate patterns. After setting the initial conditions of the cells, the basic mechanism of pattern formation is cell division and relaxation of cell positions. The results are particularly effective for big cats and giraffes.

2.6 Cellular Texture Generation

While we address a completely different a problem, our fluid animation method is inspired in part by the methodology of Fleischer et al. [21] for modelling thorns, scales, feathers, and similar surface elements whose position, orientation, and other characteristics are interdependent. Their method simulates the evolution of “cellular particles” in an “extracellular environment”. A cellular particle has a set of properties including position, orientation, radius, flags for splitting and death, and concentrations of chemicals within the cell and on the cell membrane. The extracellular environment is what a cell can “sense” and consists of concentrations of

chemicals in neighboring cell membranes, values of the implicit function defining the surface and its gradient, concentrations of diffusing chemicals and their gradients, and the difference from the average orientation of neighboring cells.

The behaviour of cells is governed by “cell programs”, which define derivatives of cell state variables. All the cell programs for a cell are superposed (summed) and the resulting system of differential equations is solved using a piecewise ordinary differential equation solver in order to handle the discontinuities. Constraints can be written as cell programs by formulating them as energy functions to be minimized.

After running the cellular particle simulator, the particles and their environment are converted to geometric and other information for rendering. Separation of the cellular particle simulation from the actual geometry (and other rendering information) has the advantage of allowing post-simulation modification of the appearance or representation of the cells based on relative image size and other factors. The disadvantage is that the actual geometry is not used in the simulation so “undesirable intersections” may be generated. Additional details can be found in Fleischer’s dissertation [20].

2.7 Summary

The state of the art in fluid animation has advanced rapidly in recent years as the effectiveness of using techniques adapted from computational fluid dynamics for numerically solving the Navier-Stokes equations in three dimensions has become clear. Recent research has presented general methods for animating incompressible, viscous, free-surface liquid flow with internal boundaries and objects moving in the flow [22] and animating incompressible, thermally buoyant gaseous flow also with internal boundaries and moving objects [19].

Still many challenges remain. For example, despite their similarity, it is not clear how to integrate the methods just mentioned to model the interaction of a gas and liquid. Chemical reactions in fluids have been modelled in an ad hoc manner for the animation of combustion and the modelling of weathering effects, but no research (to the author's knowledge) has addressed the general modelling of chemical reactions in fluids. Another challenge—perhaps the most pressing one—is control: getting flow simulation to do what you want.

Chapter 3

Framework

In this chapter we present the conceptual framework of our methodology for the computer animation of reactive fluids. This framework consists of an integrated model of fluids containing reactive chemical species and a basic algorithm for animating the various parameters of this model. Our fluid model is specifically designed for computer animation and provides a coherent basis for the integration of fluid flow simulation with both the procedural animation of chemical reactions and the representation of fluid shape and shading.

3.1 An Integrated Fluid Model

In our framework we define a fixed spatial domain $\Omega \subset \mathbb{R}^3$ to be entirely filled with matter consisting of different material types. Materials here are defined to be immiscible, i.e., materials cannot be mixed, and thus exactly one material exists at each point $\mathbf{x} \in \Omega$. A material is defined to be entirely in gas, liquid, or solid phase, and thus every point $\mathbf{x} \in \Omega$ is entirely in one of these phases. Solid materials here are used simply to define static boundaries for fluid (gas and liquid) materials,

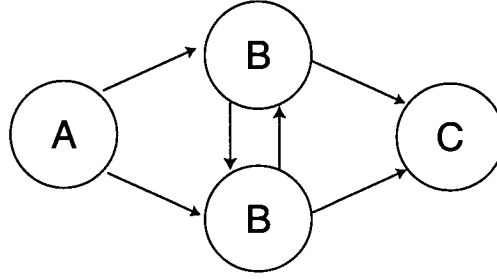


Figure 3.1: Digraph showing relationship between independent (**A**), interdependent (**B**), and dependent (**C**) parameters of our material model.

though our conceptual framework could easily be extended to address dynamic solids including various effects of fluids on solids. However, the scope of this dissertation is limited to fluids, and we will focus entirely on fluid materials.

Our material model consists of a set of parameters each of which we abstractly characterize as either *independent*, *interdependent*, or *dependent*. If we use a directed graph to represent dependency relationships between parameters, then independent parameters are nodes **A** with an in-degree of zero, interdependent parameters are nodes **B** that are part of some cycle, and dependent parameters are nodes **C** with an in-degree of at least one that are not part of any cycle (see Figure 3.1). This abstract characterization of parameters of our material model helps define appropriate techniques for their animation. Our general strategy is to use interactive and procedural animation techniques for independent parameters, dynamic simulation for interdependent parameters, and parametric models for dependent parameters.

All parameters of our material model (other than material constants) are defined over the spatial domain $\Omega \subset \mathbb{R}^3$ and the temporal domain $\mathcal{T} \subset \mathbb{R}$ as either scalar fields ($S : \Omega \times \mathcal{T} \rightarrow \mathbb{R}$) or vector fields ($\mathbf{V} : \Omega \times \mathcal{T} \rightarrow \mathbb{R}^3$). Using fields as a unified representation facilitates the integration of different modelling, simula-

| Physical | |
|--------------------------------|--|
| $\mathbf{v}(\mathbf{x}, t)$ | velocity (m/s) |
| $p(\mathbf{x}, t)$ | pressure (N/m ²) |
| $T(\mathbf{x}, t)$ | temperature (K) |
| $\mathbf{b}(\mathbf{x}, t)$ | body forces (N) |
| $C_i(\mathbf{x}, t)$ | concentration of chemical species i |
| $Q_i(\mathbf{x}, t)$ | production rate of chemical species i |
| $L_i(\mathbf{x}, t)$ | loss rate of chemical species i |
| $\lambda_i(\mathbf{x}, t)$ | diffusion coefficient of species i |
| $\Phi(\mathbf{x}, t)$ | material type (see Table 3.2) |
| Control | |
| $\alpha_C(\mathbf{x}, t)$ | degree of flow control ($\alpha_C \in [0, 1]$) |
| $\mathbf{v}_C(\mathbf{x}, t)$ | control velocity |
| $\mathcal{S}_i(\mathbf{x}, t)$ | implicit function for control surface i |
| $\mathcal{V}_i(\mathbf{x}, t)$ | density field of control volume i |
| Shape and Shading | |
| $\mathcal{L}(\mathbf{x}, t)$ | implicit function for liquid surface |
| $\mathcal{M}(\mathbf{x}, t)$ | volume density of participating media |
| $\mathcal{T}_a(\mathbf{x}, t)$ | volume texture coordinate |
| $\mathcal{T}_b(\mathbf{x}, t)$ | volume texture coordinate |
| $\mathcal{T}_c(\mathbf{x}, t)$ | volume texture coordinate |

Table 3.1: Spatially varying parameters of our material model ($\mathbf{x} \in \Omega$).

tion, and animation techniques. Table 3.1 lists all the parameters of our material model. These parameters will be described in detail in subsequent chapters; here we summarize their motivation.

Modelling physical properties of fluids such as density, pressure, temperature, and velocity is of course essential for physically based dynamic simulation. We motivate these physical parameters and derive our dynamic model for them from the Navier-Stokes equations in Chapter 4. Note that our model defines all fluids to contain the same set of arbitrary (real or hypothetical) chemical species with normalized concentrations: each chemical species i has concentration $C_i(\mathbf{x}, t) \in [0, 1]$. Thus, every point $\mathbf{x} \in \Omega$ can contain some amount of every chemical species. This

| | |
|-------------|-------------------------------|
| ϕ | phase (gas, liquid, or solid) |
| ρ | density (kg/m ³) |
| μ | viscosity |
| λ_T | thermal diffusivity |
| n_ι | index of refraction |

Table 3.2: Material constants.

facilitates the modelling of chemical reactions between different materials. Our dynamic simulation method is described in Chapter 5. A key aspect of our framework is that we integrate parameters for simulation control into our material model. These parameters include a “control velocity” that is used to “direct” the flow simulation to “perform” as desired. This method and its integration with our numerical simulation algorithm is discussed in Chapter 6. Other “control” parameters define abstract surfaces and volumes for “shaping” the simulation. These parameters are discussed in Chapter 8. The final set of parameters describes the shape and shading of liquid surfaces and participating media. Liquid surfaces are represented using implicit surfaces. Participating media are represented using volume densities. We define shading models with volume parameters. This approach facilitates the coupling of shape and shading parameters with physical properties and control parameters. Our shape and shading models and their parameters are described in Chapter 7.

3.2 Reactive Elements

In addition to our model of fluid materials, our framework defines abstract modelling primitives we call *reactive elements*. These primitives are ellipsoidal volumes with a set of associated properties that serve three distinct roles in our animation methodology. The first role is the tracking of material properties. Since we use fixed, fairly

| <i>Symbol</i> | <i>Property</i> |
|---------------|---|
| Q_i | production rate of chemical species i |
| L_i | loss rate of chemical species i |
| λ_i | diffusion coefficient of chemical species i |
| q | production rate of heat |

Table 3.3: Programmable reactive element properties.

coarse grids in our numerical simulation algorithm, it is hard to track sharp interfaces in the flow as numerical dissipation tends to smooth out such interfaces. Thus, interface tracking generally requires special techniques. We use ellipsoidal primitives to track the evolution of material properties such as density and viscosity.

The second role of reactive elements is to facilitate the interactive visualization of unsteady flow fields. Simulation-based fluid animation is a trial-and-error process that (to be practical) requires previewing flows before committing to the final rendering. While there are a variety of techniques for the visualization of unsteady, three-dimensional flow fields, the task remains challenging and is an active area of research. Moreover, these techniques are designed to facilitate the understanding of the character of the flow and not the appearance of fluids moving in it. We have found that an effective approach to “previewing” our simulated flows is to generate simple geometric models from elements that can be rendered quickly using graphics hardware. These geometric models can be appropriately shaded for additional visual cues about fluid properties.

The third role reactive elements play in our methodology is one of simulation control: elements are *programmable*. The programmable element properties are listed in Table 3.3. While element motion is governed by the fluid flow simulation, these properties can be arbitrarily animated. We define *reactive element programs* as procedures that are executed at every time step in our simulation algorithm to

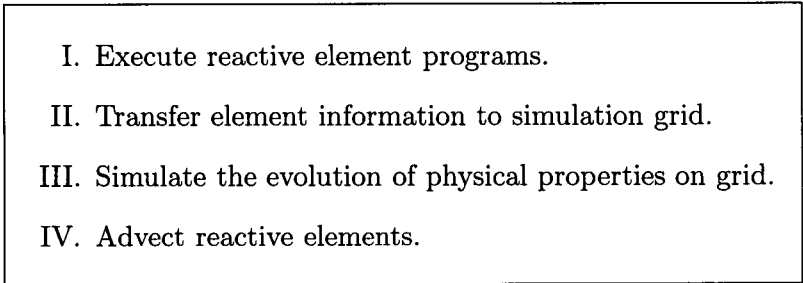
- 
- I. Execute reactive element programs.
 - II. Transfer element information to simulation grid.
 - III. Simulate the evolution of physical properties on grid.
 - IV. Advect reactive elements.

Figure 3.2: Main steps in one iteration of the simulation algorithm.

update these properties. These procedures are programs in a dedicated high level language described in Chapter 8. This procedural animation method facilitates the modelling of chemical and thermal reactions and their effects on the appearance and behaviour of our simulated fluids.

3.3 The Basic Algorithm

Our animation methodology is based on a simulation algorithm that iteratively advances through time. Before running the simulation, the animator must set the desired time step (or equivalently the desired frame rate) and model appropriate initial and boundary conditions (often simple default values are sufficient). Additionally, the user can model control surfaces and volumes, the control velocity, various material types, reactive element programs associated with each material type, and the initial state and distribution of reactive elements.

Our simulation algorithm is limited by a stability condition that restricts the maximum time step that may be taken. Thus, multiple simulation iterations may be required to advance from one frame to the next. After determining the appropriate simulation time step δt (which will be less than or equal to the inverse of the desired frame rate), one iteration of the simulation consists of the four main steps shown

in Figure 3.2. This basic algorithm will be fleshed out in Chapters 5 and 6. At the end of the one or more simulation iterations required to advance the system to the time of the next frame, the rendering algorithm discussed in Chapter 7 is used to synthesize the frame from the current state of the system.

3.4 Summary

We have presented a simple, general, and coherent framework for our animation methodology. We use an integrated model of the shape, shading, and physical properties of fluid where all parameters of this model are represented using fields. These parameters are time-evolved from initial and boundary conditions using coupled dynamic simulation and procedural animation methods.

Our framework facilitates the integrated animation of the appearance and behaviour of multiple reactive fluids. A number of challenges must be addressed to develop an effective animation method based on this framework. First among these is that we need a physically based dynamic model of fluid suitable for computer animation. Defining such a model is the subject of the next chapter.

Chapter 4

Dynamic Model

In this chapter we review the basic principles and equations of fluid dynamics. We derive our dynamic model for fluid flow from these equations by making a number of simplifying assumptions that we argue are appropriate for computer animation. We formulate this model as an initial-boundary value problem.

4.1 Introduction

In 1755 the Swiss mathematician Leonhard Euler derived a nonlinear set of partial differential equations that describe the behavior of an idealized, non-viscous fluid. From Euler's equations, the equations for viscous flow were derived by the French engineer Claude Louis Marie Henri Navier in 1822. The Irish mathematician George Gabriel Stokes rederived the equations in 1845 using a more sound theoretical basis. These equations, now known as the Navier-Stokes equations, remain the mathematical basis of all modern fluid dynamics, and the field of computational fluid dynamics is devoted to their numerical solution.

For computer animation, there is no *a priori* justification for modelling the

underlying physics of fluid flow: the sole interest is the *appearance* of flowing fluids. Indeed, as discussed previously, many effective non-physical fluid animation methods have been developed, e.g., ad hoc models of ocean waves [29, 58, 77]. However, recent work has shown that the numerical solution of the Navier-Stokes equations in three dimensions can be an effective method of realistically animating a much more general range of fluid motions [19, 22, 23, 26, 68, 89].

This dissertation addresses a larger class of fluid motion than has been considered before in computer graphics—the flow of multiple reactive fluids—and the goal of this chapter is to derive a suitable dynamic model for these flows specifically designed for the needs of computer animation. Ideally, this model should be general enough to describe all the visually important characteristics of our class of fluid motions, yet simple enough to be numerically solved using fast and robust methods. We argue that when this ideal cannot be achieved, the best approach is to simplify the dynamic model to ensure fast and robust simulation and use ad hoc techniques to achieve the desired visual effects.

4.2 Basic Principles of Fluid Flow

In this section we briefly review some basic concepts of fluid flow that are essential to understanding this dissertation. For details, the reader is referred to any good book on fluid dynamics such as the classic by Batchelor [3].

4.2.1 Fluid

At a molecular level, the phases of matter can be understood in terms of molecular kinetic energy and intermolecular forces. In solids, molecules are held closely packed and in relatively fixed positions by intermolecular forces. In fluids, molecules have

| <i>Fluid</i> | <i>Density (kg/m³)</i> |
|----------------|-----------------------------------|
| Hydrogen | 0.0899 |
| Helium | 0.1785 |
| Air | 1.293 |
| Oxygen | 1.429 |
| Carbon dioxide | 1.977 |
| Ethyl alcohol | 799 |
| Water | 1,000 |
| Glycerine | 1,260 |
| Mercury | 13,550 |

Table 4.1: Densities ρ of common fluids at 20°C and one atmosphere.

greater kinetic energy and can overcome attractive intermolecular forces to move freely past each other. Both gases and liquids are fluids. In liquids, molecules move freely but are held closely packed. In gases, molecules have enough kinetic energy to escape the attraction of surrounding molecules. The phase of a substance of course depends on its temperature T , a measure of the average molecular kinetic energy.

At a macroscopic scale—obviously the scale of interest for computer graphics—fluids are usually assumed to be continuous rather than collections of discrete molecules. At this scale, a fluid is defined in terms of stresses. A *stress* is a force per unit area acting on an infinitesimal surface element, and a *shear stress* is the stress component acting tangentially to this surface. A *fluid* is defined as a substance that cannot resist shear stress at rest. A fluid responds to even the smallest shear stress with a continuous deformation as layers of fluid slide past each other: it *flows*.

4.2.2 Density

Fluid density ρ , the mass of material per unit volume, depends on both temperature and pressure. Table 4.1 lists the densities of some common fluids (both gases and liquids) at fixed temperature and pressure values. Note that the liquids are roughly

| <i>Fluid</i> | <i>Approximate Viscosity (centipoise)</i> |
|---------------|---|
| Hydrogen | 0.0086 |
| Air | 0.0182 |
| Water | 1.002 |
| Ethyl alcohol | 1.2 |
| Whole milk | 2.12 |
| Olive oil | 84.0 |
| Pancake syrup | 2,500 |
| Honey | 10,000 |
| Peanut butter | 250,000 |
| Window putty | 100,000,000 |

Table 4.2: Approximate viscosities μ of common fluids at 20°C.

1000 times as dense as the gases.

4.2.3 Viscosity

Viscosity μ is a measure of the resistance of a fluid to flow. This resistance is caused by internal friction: molecular cohesion and momentum transfer. Viscosity (also called the *dynamic viscosity*) is defined as the ratio of the shear stress to the shear rate. The shear rate is a measure of the fluid deformation in response to a shear stress and is described by the gradient of the fluid velocity. If the viscosity is constant (for a given temperature) regardless of the shear rate (i.e., the shear rate is proportional to the shear stress), then the fluid is called *Newtonian*. Otherwise, the viscosity depends on the shear rate, and the fluid is called *non-Newtonian*. The SI unit of viscosity is (N·s/m²) or (kg/m·s). However, the Poise is more commonly used, especially the centipoise as the viscosity of water at room temperature is roughly one centipoise. Table 4.2 lists the viscosities of some common fluids. The ratio of the (dynamic) viscosity to density is known as the *kinematic viscosity*, ν ,

$$\nu = \frac{\mu}{\rho}$$

The SI unit of kinematic viscosity is (m^2/s).

The viscosity of a fluid depends on its temperature. For a gas, viscosity increases with increasing temperature since the resistance to shear stress in a gas is largely due to the transfer of molecular momentum, i.e., the drag between slower and faster layers of gas. For a liquid, viscosity decreases with increasing temperature as the resistance is dominated by cohesive forces which weaken as the kinetic energy of liquid molecules increases. The effects of pressure on viscosity are negligible except for extremely high pressures.

4.2.4 Eulerian and Lagrangian Descriptions of Fluid Motion

In a *Lagrangian* description of fluid flow, the motion of individual fluid particles is tracked and fluid properties are described with respect to these moving particles. In an *Eulerian* description, fluid properties are described with respect to fixed locations in space. Our fluid model is based on an Eulerian description of fluid flow where fluid properties are represented by scalar and vector fields. For example the fluid velocity \mathbf{v} is a vector field:

$$\mathbf{v}(x, y, z, t) = u(x, y, z, t)\mathbf{i} + v(x, y, z, t)\mathbf{j} + w(x, y, z, t)\mathbf{k}$$

where \mathbf{i} , \mathbf{j} , and \mathbf{k} are the unit vectors in the x -, y -, and z -directions respectively. Changes in fluid properties in an Eulerian frame are given by the *Stokes derivative* (also called the *total derivative* and *material derivative*):

$$\frac{D}{Dt} \equiv \frac{\partial}{\partial t} + (\mathbf{v} \cdot \nabla)$$

where, for example, the fluid acceleration is given by:

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v}.$$

| | |
|------------------|--------------------------------------|
| $Ma < 0.3$ | incompressible flow |
| $0.3 < Ma < 0.8$ | subsonic flow, no shock waves |
| $0.8 < Ma < 1.2$ | transonic flow, shock waves |
| $1.2 < Ma < 3.0$ | supersonic flow, no subsonic regions |
| $3 < Ma$ | hypersonic flow |

Table 4.3: Incompressible and compressible gaseous flow regimes.

4.2.5 Compressible and Incompressible Flow

All fluids are *compressible*: fluid density changes with applied pressure. Liquids, however, are very hard to compress, e.g., the density of water changes less than 0.5% under 100 atmospheres of applied pressure at constant temperature. In computational fluid dynamics, liquids are almost always assumed to be *incompressible*. If fluid density is assumed to be everywhere constant ($\rho \equiv c$), then naturally the flow is incompressible. Constant density, however, is not a necessary condition for incompressible flow. A sufficient condition for incompressibility is:

$$\frac{\partial \rho}{\partial t} + (\mathbf{v} \cdot \nabla) \rho = 0.$$

Thus, an incompressible flow can have variable density as long as the density is constant along streamlines.

Gases are readily compressible. However, gaseous flow is virtually incompressible unless the gas is confined to a closed container or moving at a high speed. The relative speed of a flow can be expressed in terms of the *Mach number* (Ma): the ratio of the fluid speed to the speed of sound in the fluid. As a reference point, the speed of sound in air is approximately 344 m/s (≈ 771 m.p.h.) at 21°C. Compressibility effects are generally assumed to be negligible for gaseous flows at speeds of less than $0.3 Ma \approx 100$ m/s (231 m.p.h.) for air at 21°C. Indeed, gaseous flows are typically categorized by Mach number into incompressible and compressible flow

| | |
|-----------------------------|---|
| $\rho(\mathbf{x}, t)$ | density (kg/m ³) |
| $p(\mathbf{x}, t)$ | pressure (N/m ²) |
| $T(\mathbf{x}, t)$ | temperature (K) |
| $\mathbf{v}(\mathbf{x}, t)$ | velocity (m/s) |
| $\mu(\mathbf{x}, t)$ | viscosity (centipoise) |
| $\mathbf{b}(\mathbf{x}, t)$ | body forces (N) |
| $C_i(\mathbf{x}, t)$ | concentration of chemical species i |
| $Q_i(\mathbf{x}, t)$ | production rate of chemical species i |
| $L_i(\mathbf{x}, t)$ | loss rate of chemical species i |
| $\lambda_i(\mathbf{x}, t)$ | diffusion coefficient of species i |

Table 4.4: Physical parameters of our fluid model.

regimes as shown in Table 4.3.

4.3 Dynamic Model

In our conceptual framework, the entire spatial domain $\Omega \subset \mathbb{R}^3$ is filled with fluids and solids. Each fluid consists of some material type. All material properties are defined over Ω ; the physical properties are summarized in Table 4.4. Our *dynamic model* is a mathematical description of how these physical properties change over time. This model is based on the Navier-Stokes equations. Before deriving our model from these equations, we first make a number of simplifying assumptions. For details on the equations of fluid mechanics, we refer the interested reader any good book on the subject such as the one by Aris [2].

4.3.1 Assumptions

We make several simplifying assumptions about our flows that we argue are appropriate for our animation method.

Newtonian Fluids

We assume our fluids are all Newtonian (the viscosity is independent of the shear rate) as this simplifies the numerical simulation and applies to many common fluids. This assumption, however, means that we do not fully address the behaviour of interesting non-Newtonian fluids such as shear-thinning fluids such as paint, shampoo, and ketchup as well as shear-thickening fluids such as wet sand.

Incompressible Flow

We assume that all our flows (gaseous and liquid) are incompressible. As we shall see in the next section, this assumption simplifies the equations of fluid mechanics. It also facilitates a coherent numerical solution method for these equations and is a necessary assumption for our simulation control method discussed in Chapter 6. The assumption of incompressible flow for liquids is an excellent one. For gases, it bounds the maximum flow speed that our animation method can address in a realistic fashion to 0.3 Mach, but this encompasses most common flows. Also, the compression of gases that occurs under applied pressure in closed containers is not addressed by our model.

Inert Flow

While we model the transport and reactions of chemicals in our flows, we do not explicitly model the effects of chemical reactions on the flow. For the purposes of our dynamic model, we assume chemical reactions have no effect on the fluid density, temperature, or any aspect of the flow. Instead, we present in Chapter 8 a procedural animation method for the ad hoc modelling of reactive effects on the flow. We adopt this approach for several reasons:

- The numerical simulation of reactive flow is computationally expensive. Rapid reaction rates result in a system of very stiff differential equations that must be coupled with the equations of fluid flow. The numerical solution of the resulting system is quite difficult and typically requires very small time steps.
- The tightly coupled and highly nonlinear system is inherently difficult to control for animation purposes. Even the smallest changes in the flow can trigger unwanted chemical reactions that completely alter the flow. The extremely unpredictable nature of the coupled system makes it intrinsically difficult to determine what conditions are required to generate flows with specific appearances and behaviours.
- The diversity of chemical reactions, especially the diversity of their macroscopic effects, motivates an open-ended approach focused on how they affect the appearance and behaviour of fluid flow rather than accurate chemical kinetics.

The drawback of this approach, of course, is that the burden is entirely on the animator to model the effects of chemical reactions on the flow. In Chapter 8 we introduce a programming language designed to make this task as easy as possible.

Given our assumptions, we now derive the equations of our dynamic model from the general form of the Navier-Stokes equations.

4.3.2 Conservation of Mass

Mass is neither created nor destroyed in the flows considered here, so the time rate of change in mass m is zero:

$$\frac{dm}{dt} = 0.$$

The mass of a finite volume V of fluid is the integral of the fluid density over that volume:

$$m = \int_V \rho(\mathbf{x}, t) dV.$$

Reynolds' transport theorem states that for a function $f(\mathbf{x}, t)$ and a closed volume $V(t)$ moving with the fluid:

$$\frac{D}{Dt} \int_{V(t)} f(\mathbf{x}, t) dV = \int_{V(t)} \left(\frac{df}{dt} + \nabla \cdot (f\mathbf{v}) \right) dV.$$

Applying the transport theorem to fluid mass gives:

$$\frac{dm}{dt} = \int_{V(t)} \left(\frac{d\rho}{dt} + \nabla \cdot (\rho\mathbf{v}) \right) dV = 0.$$

Since this is true for an arbitrary volume, the integrand vanishes giving the continuity equation expressing the conservation of mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\mathbf{v}) = 0, \tag{4.1}$$

which can be rewritten by expanding the divergence term as:

$$\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot (\nabla \rho) + \rho(\nabla \cdot \mathbf{v}) = 0.$$

Using our assumption that the flow is incompressible, i.e., that

$$\frac{\partial \rho}{\partial t} + (\mathbf{v} \cdot \nabla) \rho = 0,$$

equation (4.1) reduces to the continuity equation for incompressible flow:

$$\boxed{\nabla \cdot \mathbf{v} = 0} \quad (4.2)$$

which simply says that the velocity field is *solenoidal*, i.e., it has zero divergence.

4.3.3 Conservation of Momentum

The conservation equation for momentum is Newton's second law of motion which states that changes in momentum are equal to the applied forces:

$$\frac{d(m\mathbf{v})}{dt} = \sum \mathbf{f}. \quad (4.3)$$

The forces acting on a fluid include surface forces (e.g., pressure and internal friction) and body forces (e.g., gravity and electromagnetic forces). For fluids, equation (4.3) takes the form:

$$\frac{D}{Dt} \int_{\Omega} \rho \mathbf{v} = \int_S \mathbf{T} \cdot \mathbf{n} dS + \int_{\Omega} \rho \mathbf{b} d\Omega \quad (4.4)$$

where \mathbf{T} is the stress tensor, and $\mathbf{b}(\mathbf{x}, t)$ is the body force density per unit volume. The first term on the right hand side of equation (4.4) corresponds to surface forces and the second term to body forces. For our Newtonian fluids, the stress tensor is:

$$\mathbf{T} = -(p + \lambda \nabla \cdot \mathbf{v}) \mathbf{I} + 2\mu \mathbf{D},$$

where \mathbf{I} is the unit tensor, p is the static pressure, and \mathbf{D} is the rate of strain (deformation) tensor :

$$\mathbf{D} = \frac{1}{2} (\nabla \mathbf{v} + (\nabla \mathbf{v})^T).$$

Applying the transport theorem and the divergence theorem to equation (4.4) gives the following form of the momentum equation:

$$\frac{\partial(\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \mathbf{v}) = \nabla \cdot \mathbf{T} + \rho \mathbf{b} \quad (4.5)$$

which after plugging in the stress tensor \mathbf{T} gives:

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla \cdot (\rho\mathbf{v}\mathbf{v}) = -(\rho\mathbf{v})\nabla \cdot \mathbf{v} - \nabla p + (\mu + \lambda)\nabla(\nabla \cdot \mathbf{v}) + \nabla \cdot 2\mu\mathbf{D} + \rho\mathbf{b} \quad (4.6)$$

With the assumption of incompressible flow, we know that the velocity has zero divergence, i.e., that

$$\nabla \cdot \mathbf{v} = 0,$$

and thus equation (4.6) reduces to:

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla \cdot (\rho\mathbf{v}\mathbf{v}) = \nabla \cdot 2\mu\mathbf{D} - \nabla p + \rho\mathbf{b} \quad (4.7)$$

which is the momentum equation for the viscous, variable-density, variable-viscosity, incompressible flow of a Newtonian fluid. Using the product rule to expand the terms on the left hand side of equation (4.7) gives:

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \frac{\partial \rho}{\partial t} + \mathbf{v} \nabla \cdot (\rho\mathbf{v}) + \rho\mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot 2\mu\mathbf{D} - \nabla p + \rho\mathbf{b}. \quad (4.8)$$

Applying equation (4.1) to equation (4.8) gives:

$$\boxed{\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \frac{1}{\rho} (\nabla \cdot 2\mu\mathbf{D} - \nabla p) + \mathbf{b}} \quad (4.9)$$

which gives the momentum equation (4.7) in explicit terms of the fluid acceleration.

4.3.4 Conservation of Energy

The first law of thermodynamics expresses the conservation of energy leads to the *energy equation*. We use a simple form of the energy equation that describes the convection and diffusion of temperature:

$$\boxed{\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \lambda_T \nabla^2 T + q} \quad (4.10)$$

where λ_T is the *thermal diffusivity* and q is a heat source. In terms of effects of temperature on the flow, we only consider *thermal buoyancy* forces and incorporate them into the body force \mathbf{b} as follows:

$$\mathbf{b} = \mathbf{f} + (1 - \beta(T - T_\infty))\mathbf{g} \quad (4.11)$$

where T_∞ is the ambient temperature, β is a scaling factor, \mathbf{g} is gravity, and \mathbf{f} is all other body forces. Note that if the fluid temperature is uniform and equal to the ambient temperature, then the thermal buoyancy is zero.

4.3.5 Chemical Transport and Reactions

Our fluid model defines a set of chemically reactive species with concentrations $\{C_1, C_2, \dots, C_n\}$ that are transported by flow and undergo chemical reactions. The change in the concentration of species i is given by:

$$\boxed{\frac{\partial C_i}{\partial t} + \mathbf{v} \cdot \nabla C_i = \lambda_i \nabla^2 C_i + Q_i - L_i C_i} \quad (4.12)$$

where λ_i is the diffusion coefficient of species i , Q_i is the chemical production rate of species i , and L_i is the chemical loss rate of species i . The production and loss rates of a chemical species can depend on the concentrations of all other chemical species as well as all other parameters of our fluid model.

4.4 Initial-Boundary Value Problem

We can now summarize our dynamic model of fluid flow and formulate a well-defined initial-boundary problem. We model viscous incompressible fluid flow. We also model a set of chemically reactive species dissolved in the fluid but assume inert flow. We consider fluid in a domain $\Omega \subset \mathbb{R}^3$ throughout time $t \in [t_0, t_1]$ described by the variables given in Table 4.4.

4.4.1 Equations

Our dynamic model consists of the following system of partial differential equations:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \quad (4.13)$$

$$\begin{aligned} \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z} &= \frac{1}{\rho} \left[\mu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) - \frac{\partial p}{\partial x} \right] \\ &+ b_x \end{aligned} \quad (4.14)$$

$$\begin{aligned} \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + w \frac{\partial v}{\partial z} &= \frac{1}{\rho} \left[\mu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \right) - \frac{\partial p}{\partial y} \right] \\ &+ b_y \end{aligned} \quad (4.15)$$

$$\begin{aligned} \frac{\partial w}{\partial t} + u \frac{\partial w}{\partial x} + v \frac{\partial w}{\partial y} + w \frac{\partial w}{\partial z} &= \frac{1}{\rho} \left[\mu \left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \right) - \frac{\partial p}{\partial z} \right] \\ &+ b_z \end{aligned} \quad (4.16)$$

$$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + w \frac{\partial T}{\partial z} = \lambda_T \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) + q \quad (4.17)$$

$$\begin{aligned} \frac{\partial C_i}{\partial t} + u \frac{\partial C_i}{\partial x} + v \frac{\partial C_i}{\partial y} + w \frac{\partial C_i}{\partial z} &= \lambda_i \left(\frac{\partial^2 C_i}{\partial x^2} + \frac{\partial^2 C_i}{\partial y^2} + \frac{\partial^2 C_i}{\partial z^2} \right) \\ &+ Q_i - L_i C_i \end{aligned} \quad (4.18)$$

$$\frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial x} + v \frac{\partial \rho}{\partial y} + w \frac{\partial \rho}{\partial z} = 0 \quad (4.19)$$

$$\frac{\partial \mu}{\partial t} + u \frac{\partial \mu}{\partial x} + v \frac{\partial \mu}{\partial y} + w \frac{\partial \mu}{\partial z} = 0 \quad (4.20)$$

These equations are the component forms of the equations previously presented in dimensionless form. Equation (4.13) corresponds to the continuity equation (4.2), equations (4.14–4.16) correspond to the momentum equation (4.9) under the assumption of constant viscosity, equation (4.17) corresponds to the energy equation (4.10), and equation (4.18) corresponds to equation (4.12) describing chemical trans-

port and reactions. Equations (4.19) and (4.20) respectively describe the transport of density and viscosity.

4.4.2 Initial Conditions

Initial conditions at time $t = t_0$ are required for all fluid properties given in Table 4.4. Since our methodology encompasses various control techniques, it is generally easier to use simple default values for these conditions and use the control techniques to drive the simulation towards the desired starting state of the scene at some intermediate time $t_i \in [t_0, t_1]$.

4.4.3 Boundary Conditions

In addition to initial conditions, conditions on the boundary Γ of the domain Ω are required for all times $t \in [t_0, t_1]$. Velocity values are subject to the kinematic mass-conservation constraint (4.13), and thus velocity on the boundary must satisfy:

$$\int_{\Gamma} \mathbf{v} \cdot \mathbf{n} d\Gamma = 0$$

which says that the flow coming into Ω must always equal the flow going out.

The next chapter discusses the numerical solution of this initial-boundary value problem.

Chapter 5

Numerical Simulation

In this chapter, we describe the numerical solution of the initial-boundary value problem defined in the previous chapter. The continuous partial differential equations are discretely approximated using finite differences, converting them to a system of linear algebraic equations. We then present our solution algorithm based on a Chorin projection scheme.

5.1 Introduction

Our initial-boundary value problem describes the unsteady, viscous, incompressible flow of gases and liquids in a three dimensional domain with internal boundaries. Numerical solution of this problem presents difficult challenges and is an ongoing area of research in computational fluid dynamics. Fortunately, we are solely interested in the *appearance* of fluid flow and can tolerate relatively crude numerical approximations as long as the resulting animations appear realistic. The numerical solution itself is not of interest per se; only its *effects* on our shape and shading models are significant. In practice this means we can (and do) use fairly coarse uniform

grids to approximate fluid properties, and we can (and do) accept some numerical diffusion and other approximation errors in exchange for speed and stability.

The numerical solution of the incompressible Navier-Stokes equations has been investigated for decades in computational fluid dynamics, and a wide variety of solution methods have been developed. The goal of this chapter is to adopt appropriate methods for our animation methodology. In particular, we want rapid and robust methods that can be effectively coupled with control techniques, shape and shading models, and reactive elements.

Our simulation algorithm is based on a projection scheme [10], a standard approach to numerically solving the incompressible Navier-Stokes equations in computational fluid dynamics that has recently become the favored approach for the computer animation of fluids [19, 22, 68]. A readable description of a basic implementation of a numerical simulation algorithm for the incompressible Navier-Stokes equations in two dimensions based on a projection scheme can be found in the book by Griebel et al. [35]. Before discussing our simulation algorithm, we first discuss the discretization of the spatial domain and differential equations using finite differences.

5.2 Finite Differences

We use finite differences to discretize the continuous partial differential equations (4.13-4.20) constituting our initial-boundary value problem into a system of algebraic equations. There are a number of other well-known discretization schemes such as finite volumes and finite elements, but finite differences are straightforward and effective for the regular grid we use to discretize the spatial domain.

Finite difference approximations can be derived from Taylor series. Any continuous differentiable function $f(x)$ can be expressed as a Taylor series expansion

about x_0 :

$$f(x) = f(x_0) + (x - x_0) \left(\frac{\partial f}{\partial x} \right)_{x_0} + \frac{(x - x_0)^2}{2!} \left(\frac{\partial^2 f}{\partial x^2} \right)_{x_0} + \frac{(x - x_0)^3}{3!} \left(\frac{\partial^3 f}{\partial x^3} \right)_{x_0} + \cdots + \frac{(x - x_0)^n}{n!} \left(\frac{\partial^n f}{\partial x^n} \right)_{x_0} + h$$

where h stands for the higher order terms. Using the Taylor series expansion for $f(x)$ about x_i at $x = x_{i+1}$ gives:

$$\left(\frac{df}{dx} \right)_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{2} \left(\frac{\partial^2 f}{\partial x^2} \right)_i - \frac{(x_{i+1} - x_i)^2}{6} \left(\frac{\partial^3 f}{\partial x^3} \right)_i + h$$

Truncating this series after the first term on the right hand side gives the *forward difference* approximation to the first derivative of $f(x)$ at $x = x_i$:

$$\left[\frac{df}{dx} \right]_i^{(f)} \equiv \frac{f_{i+1} - f_i}{x_{i+1} - x_i} = \left(\frac{df}{dx} \right)_{x_i} - \epsilon_T$$

where ϵ_T is the *truncation error*, i.e., the terms deleted from the Taylor series expansion. The truncation error is usually dominated by the first truncated term.

A higher order approximation to the first derivative can be derived from taking the Taylor series expansion of $f(x)$ about x_i at both x_{i-1} and x_{i+1} :

$$\left(\frac{df}{dx} \right)_i = \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}} - \frac{(x_{i+1} - x_i)^2 - (x_i - x_{i-1})^2}{2(x_{i+1} - x_{i-1})} \left(\frac{\partial^2 f}{\partial x^2} \right)_i - \frac{(x_{i+1} - x_i)^3 - (x_i - x_{i-1})^3}{6(x_{i+1} - x_{i-1})} \left(\frac{\partial^3 f}{\partial x^3} \right)_i + h$$

Truncating this series after the first term on the right hand side gives the *centered difference* approximation to the first derivative of $f(x)$ at $x = x_i$:

$$\left[\frac{df}{dx} \right]_i^{(c)} \equiv \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}}$$

For uniform spacing, i.e., for

$$x_{i+1} - x_i = x_i - x_{i-1} = \delta x,$$

the first term in the truncation error for the centered difference approximation vanishes. Using a similar derivation gives the following approximation of the second derivative of $f(x)$ at x_i :

$$\left[\frac{d^2 f}{dx^2} \right]_i \equiv \frac{f_{i+1} - 2f_i + f_{i-1}}{\delta x^2}$$

which is second-order accurate.

5.3 Approximation of Convective Transport

A well-known challenge of Eulerian fluid flow simulation is the approximation of convective transport which is generally described by:

$$\frac{\partial a}{\partial t} = -\mathbf{v} \cdot \nabla a$$

for a scalar quantity $a(\mathbf{x}, t)$ moving with the fluid velocity $\mathbf{v}(\mathbf{x}, t)$. Naive use of finite differences can result in numerical diffusion and instability. Two common approaches for handling convective transport are upwind differencing and the donor-cell scheme. Stam [68] has shown that the use of a semi-Lagrangian integration scheme [71] can be the key to a unconditionally stable approach to solving the incompressible Navier-Stokes equations. In this scheme, convective transport is computed by simply backtracing through time along streamlines to sample values of fluid properties at the previous time step. For example, the velocity of a fluid particle at a grid point $\mathbf{x}_{i,j,k}$ at time t_{n+1} is the velocity that particle had at time t_n . All that has to be computed is the position of that particle at time t_n which is

found by following the path of that particle backwards through time. This approach ensures that convected values at time t_{n+1} are never greater than they were at time t_n . However, it introduces numerical diffusion and is generally too inaccurate for most applications in computational fluid dynamics. However, it has been shown to work well for computer animation [19, 22, 68] where this dissipation is accepted for robustness and the ability to take large time steps. We use the semi-Lagrangian integration scheme with second-order Runge Kutta integration for all convective terms unless otherwise indicated.

5.4 Discretization of the Domain

Fluid density (ρ), pressure (p), temperature (T), and velocity ($\mathbf{v} = (u, v, w)$) are defined over the three dimensional spatial domain Ω . These properties are discretely represented using a fixed, axis-aligned, uniform grid that subdivides Ω into a $N_x \times N_y \times N_z$ array of cubical *cells* with linear dimension $\delta\tau$, i.e., the grid spacing in each coordinate direction is $\delta\tau$. We define without loss of generality the origin of this grid to be at the origin of the coordinate system and thus Ω is defined as the following cuboid:

$$\Omega = [0, N_x\delta\tau] \times [0, N_y\delta\tau] \times [0, N_z\delta\tau] \subset \mathbb{R}^3,$$

We use zero-based indexing for cells, i.e., the grid consists of cells (i, j, k) where $i \in \{0, \dots, N_x - 1\}$, $j \in \{0, \dots, N_y - 1\}$, and $k \in \{0, \dots, N_z - 1\}$. Thus, cell (i, j, k) occupies the volume

$$[i\delta\tau, (i+1)\delta\tau] \times [j\delta\tau, (j+1)\delta\tau] \times [k\delta\tau, (k+1)\delta\tau]$$

The grid is “padded” with a layer of *boundary cells* that facilitate the application of boundary conditions as shown in Figure 5.1. Simulation occurs only in non-

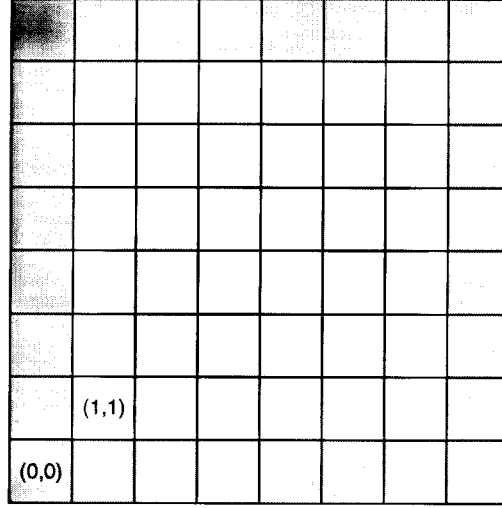


Figure 5.1: Boundary cells (darker) and computational cells (lighter). A two-dimensional grid is shown for clarity; our methodology uses a three-dimensional grid.

boundary or *computational cells*, i.e., cells (i, j, k) where $i \in \{1, \dots, N_x - 2\}$, $j \in \{1, \dots, N_y - 2\}$, and $k \in \{1, \dots, N_z - 2\}$. Computational cells are defined to be either *gas cells*, *liquid cells*, or *solid cells* depending on the phase of the material in that cell. In our implementation, no computation is done in solid cells, but this would be a natural extension of our work. By default, all cells are gas cells. If a cell contains any solid material it is classified as a solid cell. Any cell containing no solid material and any liquid material is a liquid cell. Cells are considered *neighbors* or *adjacent* if and only if they share a face: each computational cell has exactly six neighbors.

We use the *staggered grid* approach originally due to Harlow and Welch [37]. In this approach, all fluid properties except velocity are taken at cell centers, e.g., the pressure for cell (i, j, k) is:

$$p_{i,j,k} = p(\delta\tau/2 + i\delta\tau, \delta\tau/2 + j\delta\tau, \delta\tau/2 + k\delta\tau)$$

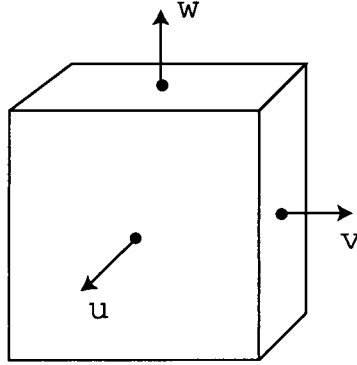


Figure 5.2: Velocity components are staggered on the centers of cell faces. All other properties are taken at cell centers.

Velocity components are taken on cell faces orthogonal to the direction of the velocity component (see Figures 5.2 and 5.3):

$$u_{i,j,k} = u(\delta\tau + i\delta\tau, \delta\tau/2 + j\delta\tau, \delta\tau/2 + k\delta\tau)$$

$$v_{i,j,k} = v(\delta\tau/2 + i\delta\tau, \delta\tau + j\delta\tau, \delta\tau/2 + k\delta\tau)$$

$$w_{i,j,k} = w(\delta\tau/2 + i\delta\tau, \delta\tau/2 + j\delta\tau, \delta\tau + k\delta\tau)$$

This staggered arrangement helps prevent artificial pressure oscillations [37].

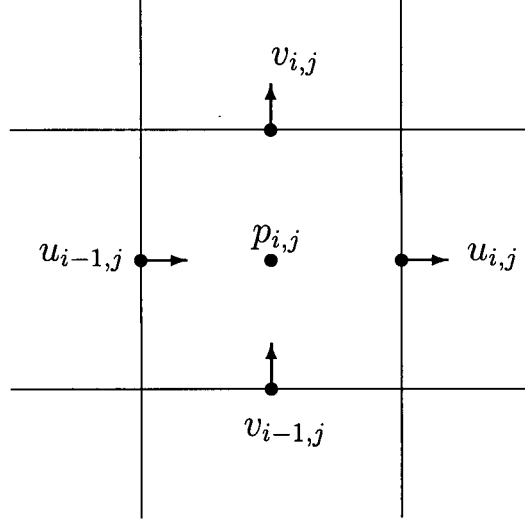


Figure 5.3: Indexing for staggered values on a two-dimensional cell (i, j) . A two-dimensional cell is shown for clarity; our methodology uses three-dimensional cells.

5.5 Discretization of the Derivatives

5.5.1 Continuity Equation

The continuity equation for incompressible flow (4.13) is discretized at the center of cell (i, j, k) using centered differences with half the cell spacing:

$$\left[\frac{\partial u}{\partial x} \right]_{i,j,k} = \frac{u_{i,j,k} - u_{i-1,j,k}}{\delta \tau} \quad (5.1)$$

$$\left[\frac{\partial v}{\partial y} \right]_{i,j,k} = \frac{v_{i,j,k} - v_{i,j-1,k}}{\delta \tau} \quad (5.2)$$

$$\left[\frac{\partial w}{\partial z} \right]_{i,j,k} = \frac{w_{i,j,k} - w_{i,j,k-1}}{\delta \tau} \quad (5.3)$$

5.5.2 Momentum Equation

The non-convective derivatives of the momentum equation (4.14) for u are discretized as:

$$\begin{aligned}\left[\frac{\partial p}{\partial x}\right]_{i,j,k} &= \frac{p_{i+1,j,k} - p_{i,j,k}}{\delta\tau} \\ \left[\frac{\partial^2 u}{\partial x^2}\right]_{i,j,k} &= \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{(\delta\tau)^2} \\ \left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j,k} &= \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{(\delta\tau)^2} \\ \left[\frac{\partial^2 u}{\partial z^2}\right]_{i,j,k} &= \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{(\delta\tau)^2}\end{aligned}$$

The approximations for the terms in the momentum equations (4.15-4.16) for v and w are analogous.

5.5.3 Energy Equation

The non-convective derivatives in the energy equation (4.17) are approximated as:

$$\begin{aligned}\left[\frac{\partial T}{\partial t}\right]_{i,j,k} &= \frac{1}{\delta t} (T_{i,j,k}^{n+1} - T_{i,j,k}^n) \\ \left[\frac{\partial^2 T}{\partial x^2}\right]_{i,j,k} &= \frac{T_{i+1,j,k} - 2T_{i,j,k} + T_{i-1,j,k}}{\delta\tau^2} \\ \left[\frac{\partial^2 T}{\partial y^2}\right]_{i,j,k} &= \frac{T_{i,j+1,k} - 2T_{i,j,k} + T_{i,j-1,k}}{\delta\tau^2} \\ \left[\frac{\partial^2 T}{\partial z^2}\right]_{i,j,k} &= \frac{T_{i,j,k+1} - 2T_{i,j,k} + T_{i,j,k-1}}{\delta\tau^2}\end{aligned}$$

5.5.4 Chemical Transport and Reactions

The non-convective derivatives in the equation for chemical transport and reactions (4.18) are approximated as:

$$\begin{aligned}\left[\frac{\partial C}{\partial t}\right]_{i,j,k} &= \frac{1}{\delta t} (C_{i,j,k}^{n+1} - C_{i,j,k}^n) \\ \left[\frac{\partial^2 C}{\partial x^2}\right]_{i,j,k} &= \frac{C_{i+1,j,k} - 2C_{i,j,k} + C_{i-1,j,k}}{\delta \tau^2} \\ \left[\frac{\partial^2 C}{\partial y^2}\right]_{i,j,k} &= \frac{C_{i,j+1,k} - 2C_{i,j,k} + C_{i,j-1,k}}{\delta \tau^2} \\ \left[\frac{\partial^2 C}{\partial z^2}\right]_{i,j,k} &= \frac{C_{i,j,k+1} - 2C_{i,j,k} + C_{i,j,k-1}}{\delta \tau^2}\end{aligned}$$

5.6 Algorithm

Our simulation algorithm time-evolves the physical properties of our fluid model by numerically solving equations (4.13-4.20). This algorithm is based on a classic projection method [10] which computes intermediate velocity values without regard for mass conservation and then corrects the velocity to ensure incompressibility. Given initial and boundary conditions, the simulation iteratively advances through time starting at time t_0 . One iteration of our simulation algorithm advancing the system from time t_n to t_{n+1} consists of the steps summarized in Figure 5.4. Steps III(A) and III(B) are computed using simple Euler integration.

The computation of the velocity is more involved, and the process can be summarized as:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \delta t \left(\mathbf{F} - \frac{1}{\rho} \nabla p^{n+1} \right)$$

where \mathbf{F} is all terms other than the pressure term in the momentum equation (4.9).

- I. Execute reactive element programs.
- II. Interpolate element information to simulation grid.
- III. Simulate the evolution of physical properties on grid.
 - A. Compute chemical reactions to give C_i^{n+1} .
 - B. Compute temperature T^{n+1} .
 - C. Compute intermediate velocity \mathbf{v}_* .
 - D. Solve Poisson equation for pressure p^{n+1} .
 - E. Correct velocity to give \mathbf{v}^{n+1} .
- IV. Advect reactive elements.

Figure 5.4: Steps in one iteration of the simulation algorithm.

Note that this is implicit in pressure. Computing \mathbf{v}^{n+1} involves two steps. First, an intermediate velocity \mathbf{v}^* that ignores the pressure term is computed:

$$\mathbf{v}^* = \mathbf{v}^n + \delta t \mathbf{F} \quad (5.4)$$

and second, the projection phase where pressure values are used to correct the velocity:

$$\mathbf{v}^{n+1} = \mathbf{v}^* - \delta t \frac{1}{\rho} \nabla p^{n+1} \quad (5.5)$$

To compute the pressure at the new time step, p^{n+1} , such that the new velocity field conserves mass, the continuity equation for incompressible flow (4.2) is applied to equation (5.5):

$$\nabla \cdot \mathbf{v}^{n+1} = \nabla \cdot \left(\mathbf{v}^* - \delta t \frac{1}{\rho} \nabla p^{n+1} \right) = 0$$

which (assuming density is constant in the given domain) gives a Poisson equation

for the pressure:

$$\nabla^2 p^{n+1} = \frac{\rho}{\delta t} (\nabla \cdot \mathbf{v}^*). \quad (5.6)$$

Solution of this equation for the pressure allows equation (5.5) to be used to correct the intermediate velocity. If both liquids and gases are present in the simulation, we first simulate liquids without regard for the gases, then simulate gases where the liquid-gas interface is taken as fixed. A more accurate approach would be to use a variable density simulation, though it is not clear if this is necessary for computer animation purposes.

5.7 Boundary Conditions

Our initial-boundary value problem requires that velocity values on the boundary Γ of the domain Ω satisfy the constraint:

$$\int_{\Gamma} v_n d\Gamma = 0$$

where $v_n(x, y, z)$ is the velocity component normal to the boundary (pointing outwards). Since our grid is axis-aligned, v_n will be equal to one of the velocity components: u , v , or w . We also define $v_t(x, y, z)$ as the velocity component tangential to the boundary. Boundary conditions must be set on the faces between fluid (gas or liquid) cells and boundary cells as well as between fluid and solid cells. We use four common boundary conditions.

5.7.1 No-Slip Boundary Conditions

At a *no-slip* boundary, there is no fluid motion relative to the boundary. For a stationary boundary, the fluid velocity should vanish, i.e.,

$$v_n = 0, \quad v_t = 0.$$

5.7.2 Free-Slip Boundary Conditions

At a *free-slip condition*, there is no flow through the boundary, but there can be fluid motion tangential to the boundary:

$$v_n = 0, \quad \partial v_t / \partial n = 0$$

where $\frac{\partial}{\partial n}$ denotes the derivative with respect to the normal v_n direction.

5.7.3 Outflow Boundary Conditions

At an *outflow boundary*, the normal derivative of the velocity vanishes, i.e., there is no change in the velocity in the direction normal to the boundary.

$$\partial v_n / \partial n = 0, \quad \partial v_t / \partial n = 0.$$

5.7.4 Periodic Boundary Conditions

For *periodic boundary conditions* (which are applicable only on the domain boundary) the velocity values on opposite sides on the domain must coincide. Since our grid is padded with a layer of boundary cells, the period length is one cell length greater than the simulation domain.

5.8 Free Surface Conditions

The free liquid surface (liquid-gas interface) is contained in liquid cells that neighbor gas cells. In these cells, the pressure must be set at cell centers to the desired gas pressure (here taken as zero) and the velocity must be set on all faces shared with gas cells so that there is zero divergence in the cell, i.e., the mass-conservation constraint (4.13) is satisfied. This constraint is generally insufficient to derive unique conditions for the “free” velocities of cells containing the liquid surface. Thus, reasonable assumptions must be made, and appropriate conditions derived for all $2^6 - 1 = 63$ configurations of gas neighbors around a liquid cell. These conditions have not been published to the author’s knowledge (and they are painful to get right), so we give explicit code for them in Appendix A. Here we briefly discuss the motivation for these free-surface conditions.

5.8.1 One Gas Neighbor (Six Configurations)

If a liquid cell has one gas neighbor, we simply set the velocity on the face between the liquid and gas cell so that there is no divergence in the liquid cell. This is the only uniquely determined case.

5.8.2 Two Gas Neighbors (15 Configurations)

If a liquid cell has two gas neighbors, these neighbor cells are either opposite each other or share an edge. If the two neighboring gas cells are opposite, then there is no unique approximation to the liquid surface normal. We assume that the fluid moves only due to body forces. If the two gas neighbors share an edge, we propagate the velocity from opposite faces and add half of the difference between the two opposite liquid faces.

5.8.3 Three Gas Neighbors (20 Configurations)

If a liquid cell has three gas neighbors, then the cell is either a "corner" (no gas neighbors are on opposite sides) or two of the gas neighbors are opposite. In the corner case the velocity values from opposite sides are propagated. In the other case, we set body forces on the two opposite gas neighbors, and set the remaining velocity value so that there is zero divergence in the cell.

5.8.4 Four Gas Neighbors (15 Configurations)

If a liquid cell has four gas neighbors, at least one pair of gas neighbors are opposite. The opposite pair (or pairs) are set to body forces. This results in either all velocity values being set or two gas neighbors of the liquid cell that share only an edge (in which case we apply the appropriate two gas neighbor conditions described previously).

5.8.5 Five Gas Neighbors (Six Configurations)

If a liquid cell has five gas neighbors, then two pairs of these neighbors must be opposite and are set according to body forces only. The remaining velocity is set to enforce zero divergence for the cell.

5.8.6 Six Gas Neighbors (One Configuration)

If all the neighbors of a liquid cell are gas, then it is assumed that the fluid motion is due entirely to body forces. The velocity on all faces is set to the old velocity plus the body force times the time step.

5.9 The Pressure Poisson Equation

Solution of the pressure Poisson equation (5.6) is a key step in our simulation algorithm and can dominate the computational effort required by our entire simulation algorithm. After the pressure values are solved for, they are used to correct the intermediate velocity \mathbf{v}^* so that the resulting velocity \mathbf{v}^{n+1} conserves mass (has zero divergence). Rewriting equation (5.6) in Cartesian coordinates gives:

$$\frac{\partial^2 p^{n+1}}{\partial x^2} + \frac{\partial^2 p^{n+1}}{\partial y^2} + \frac{\partial^2 p^{n+1}}{\partial z^2} = \frac{\rho}{\delta t} \left(\frac{\partial u^*}{\partial x} + \frac{\partial v^*}{\partial y} + \frac{\partial w^*}{\partial z} \right).$$

The terms on the left hand side of this equations can be approximated using centered differences taken at the center of a voxel as follows (dropping the time step notation):

$$\begin{aligned} \left[\frac{\partial^2 p}{\partial x^2} \right]_{i,j,k} &= \frac{p_{i+1,j,k} - 2p_{i,j,k} + p_{i-1,j,k}}{(\delta\tau)^2} \\ \left[\frac{\partial^2 p}{\partial y^2} \right]_{i,j,k} &= \frac{p_{i,j+1,k} - 2p_{i,j,k} + p_{i,j-1,k}}{(\delta\tau)^2} \\ \left[\frac{\partial^2 p}{\partial z^2} \right]_{i,j,k} &= \frac{p_{i,j,k+1} - 2p_{i,j,k} + p_{i,j,k-1}}{(\delta\tau)^2} \end{aligned}$$

and the terms for the divergence of \mathbf{v}^* on the right hand side can be approximated as shown in Section 5.5.1. Using these approximations, equation (5.6) can be approximated at the center of cell (i, j, k) by:

$$\begin{aligned} p_{i+1,j,k} + p_{i-1,j,k} + p_{i,j+1,k} + p_{i,j-1,k} + p_{i,j,k+1} + p_{i,j,k-1} - 6p_{i,j,k} = \\ \rho_{i,j,k} \frac{\delta\tau}{\delta t} (u_{i,j,k}^* - u_{i-1,j,k}^* + v_{i,j,k}^* - v_{i,j-1,k}^* + w_{i,j,k}^* - w_{i,j,k-1}^*). \end{aligned} \quad (5.7)$$

Note the Neumann boundary condition that $\partial p / \partial n = 0$ means that pressure terms across a boundary will cancel out of equation (5.7), e.g., if fluid voxel (i, j, k) has a solid neighbor voxel $(i + 1, j, k)$, then $p_{i+1,j,k} = p_{i,j,k}$ and these terms cancel each other out of equation (5.7).

5.9.1 Linear System

Equation (5.7) applies to every fluid voxel resulting in a system of linear algebraic equations:

$$A\mathbf{x} = \mathbf{b} \quad (5.8)$$

where A is a $n \times n$ matrix where n is the number of fluid voxels, \mathbf{x} is a column vector of the unknown pressure values, and \mathbf{b} is a column vector corresponding to the right hand side of equation (5.7). The diagonal coefficients a_{ii} of A are the number of fluid neighbors of voxel i , and the off-diagonal coefficients a_{ij} are nonzero if and only if fluid voxels i and j are adjacent in which case $a_{ij} = a_{ji} = -1$.

5.9.2 Iterative Solution Methods

Since the matrix A is symmetric positive definite, a number of iterative solution methods are applicable. A simple yet still practical technique (used by Foster and Metaxas [24] for enforcing incompressibility) is Successive Over-Relaxation (SOR). Unfortunately, SOR only works well for a narrow range of values of its relaxation factor, and it is generally not feasible to determine this *a priori*. The Preconditioned Conjugate Gradient (PCG) method has much better convergence properties and is our method of choice. We use an incomplete Cholesky method for preconditioning. We have found PCG to be extremely robust for solving our linear system.

5.9.3 Error Tolerance

Regardless of the iterative solution algorithm used, we need to establish an appropriate error metric and tolerance for terminating the algorithm. Two common error

metrics are the discrete L^2 -norm,

$$\|r\|_2 = \left(\frac{1}{n} \sum_{i=1}^n (r_i)^2 \right)^{\frac{1}{2}},$$

and maximum norm,

$$\|r\|_\infty = \max(|r_1|, \dots, |r_n|)$$

where $\mathbf{r} = A\mathbf{x} - \mathbf{b}$ is the residual vector of the current iteration. We use the L^2 -norm and stop iterating when

$$\|r\|_2 \leq \epsilon$$

where ϵ is the error tolerance.

5.10 Interface Tracking

We simulate multiple fluids and want to maintain sharp interfaces between them, i.e., we want to avoid numerical dissipation of material properties. Maintaining sharp interfaces with Eulerian simulation (especially with the relatively coarse grids we use) generally requires special techniques. We use ellipsoidal primitives (reactive elements) as massless markers to track fluid material properties, i.e., a form of the Marker-and-Cell (MAC) method [37]. Properties needed by the simulation such as density and viscosity are transferred from elements to the grid by weighted interpolation at the beginning of each time step. At the end of each time step, the computed velocity is used to advect these elements. As the flow can bunch elements into sparse strands, we randomly perturb (or “jitter”) elements by a small amount to maintain a more uniform distribution.

5.11 Stability Condition

Our numerical solution method is explicit in time and is *conditionally stable* subject to the well-known *Courant-Friedrichs-Levy (CFL) conditions* :

$$|u_{\max}|\delta t < \delta x$$

$$|v_{\max}|\delta t < \delta y$$

$$|w_{\max}|\delta t < \delta z$$

which state that a fluid particle may not travel a distance greater than the smallest cell dimension in a single time step. Thus, the maximum time step δt_{\max} that guarantees stability is:

$$\delta t_{\max} = \min \left(\frac{\delta x}{|u_{\max}|}, \frac{\delta y}{|v_{\max}|}, \frac{\delta z}{|w_{\max}|} \right) \quad (5.9)$$

Using a longer time step could lead to spurious oscillations that rapidly dominate the system. In our implementation, the user specifies the desired time step (or, equivalently, the desired rate of frames per simulation second). Naturally we never exceed δt_{\max} , so multiple simulation iterations per frame may be required.

5.12 Results

In this section we present the results of applying our simulation algorithm to four different animation scenarios. The first two scenarios (A and B) we assess qualitatively; the second two (C and D) we assess quantitatively. All scenarios are animated at 30 frames per second and start at frame 100 for time $t = 0$.

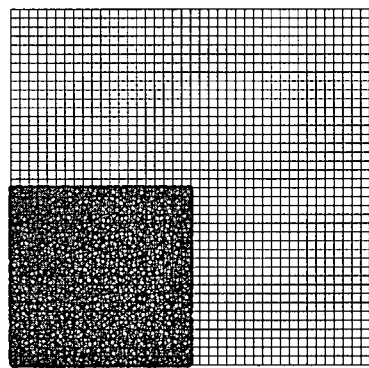
Scenario A is a classic test problem for free-surface flows, the “breaking dam” scenario where a hypothetical vertical wall holding liquid to one side of the domain

is instantly removed at time $t = 0$. Free-slip conditions are imposed on the boundary of the narrow $40 \times 3 \times 40$ simulation domain, i.e., there are 40 cells in the x -direction, 3 cells in the y -direction, and 40 cells in the z -direction. Figures 5.5–5.6 show frames from this sequence. In all examples the positive z -direction is taken to be upwards. In the views here, the positive x -axis points to the right, and the positive y -axis points into the page (a right-handed coordinate system is used). In this sequence, the liquid spills into the right side of the domain and flows up the right boundary. Note that volume is conserved because the flow is incompressible.

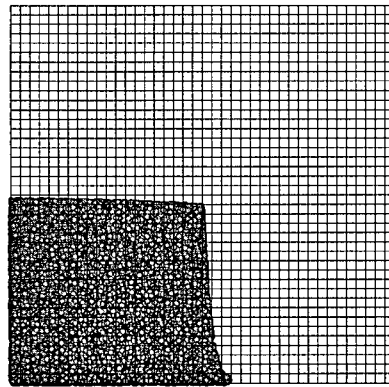
Scenario B involves a liquid body suspended in empty space (no gas is simulated) at time $t = 0$. Figures 5.7–5.8 show frames from this sequence. As the liquid crashes into the bottom of the domain, it spills up the sides where free-slip conditions are used. The elements here do not just visualize the flow but also serve to track the liquid: liquid cells are defined as those that contain elements in liquid phase. In Chapter 7, a rendered sequence corresponding to the preview images here is shown in Figures 7.1–7.2.

To analyze the convergence and other properties of our simulation algorithm, we now consider two additional scenarios in more detail. Scenario C consists of an axis-aligned cubical spatial domain with size $10 \times 10 \times 10$ reference units (nominally 0.25 meter) in the x , y , z directions respectively. Free-slip boundary conditions are used. A liquid body with dimensions $5 \times 5 \times 9$ reference units is suspended in the centre of this domain. This liquid has a viscosity of 10 centipoise (somewhat more viscous than water). Scenario C was run at various resolutions for 50 frames (starting at frame 100):

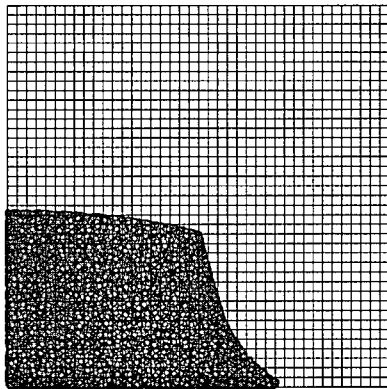
- grid resolution $10 \times 10 \times 10$, i.e., 10 cells in the x , y , z directions respectively, with grid spacing $\delta\tau = 1.0$ and 16,362 liquid elements initially in 360 liquid



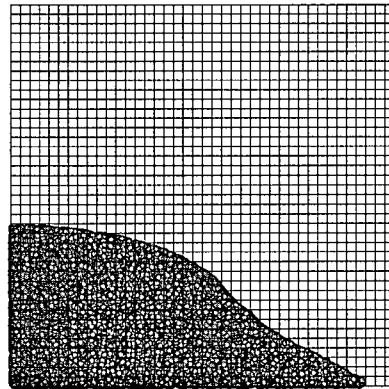
$t = 0$



$t = 0.5$

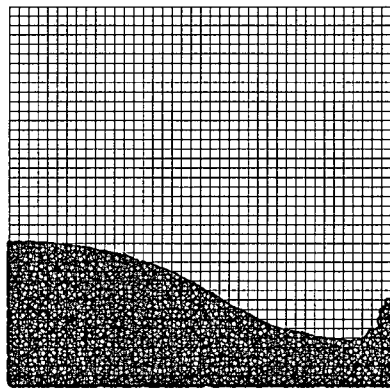


$t = 1.0$

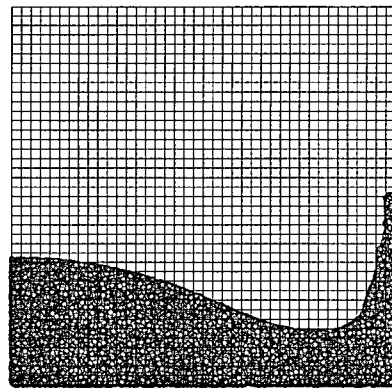


$t = 1.5$

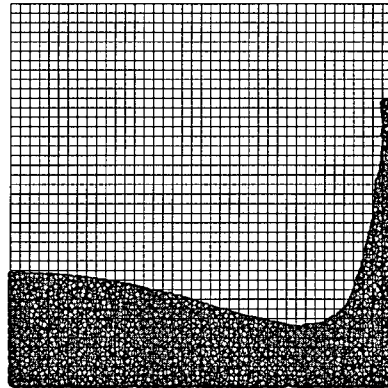
Figure 5.5: First set of frames from Scenario A (dam break).



$t = 2.0$

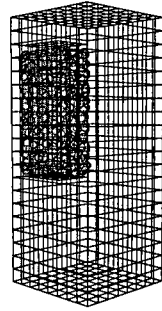


$t = 2.5$

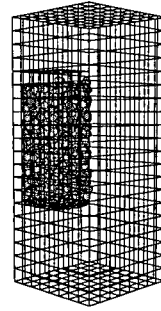


$t = 3.0$

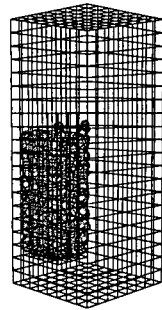
Figure 5.6: Second set of frames from Scenario A (dam break).



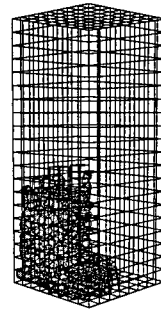
$t = 0$



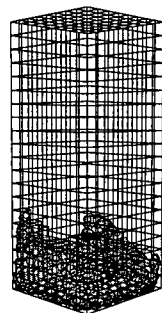
$t = 0.33$



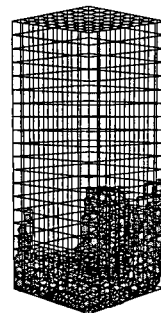
$t = 0.67$



$t = 1.0$

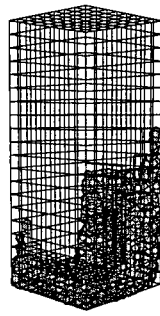


$t = 1.33$

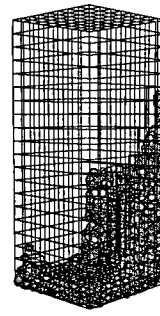


$t = 1.67$

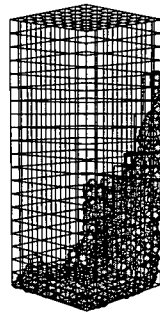
Figure 5.7: First set of frames from Scenario B.



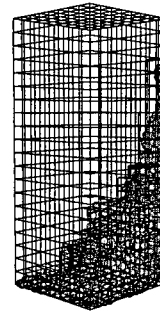
$t = 2.0$



$t = 2.33$



$t = 2.67$



$t = 3.0$

Figure 5.8: Second set of frames from Scenario B.

cells with 1,000 total cells,

- grid resolution $20 \times 20 \times 20$ with $\delta\tau = 0.5$ and 122,758 liquid elements initially in 2,299 liquid cells with 8,000 total cells,
- grid resolution $30 \times 30 \times 30$ with $\delta\tau = 0.333$ and 405,634 liquid elements initially in 7,168 liquid cells with 27,000 total cells, and
- grid resolution $40 \times 40 \times 40$ with $\delta\tau = 0.25$ and 951,390 liquid elements initially in 16,317 liquid cells with 64,000 total cells.

Figure 5.9 shows several frames from the $20 \times 20 \times 20$ resolution run of Scenario C.

Scenario D is similar but the spatial domain has size $50 \times 50 \times 50$ reference units and the liquid body has size $25 \times 25 \times 45$ reference units. Scenario D was run at the following resolutions for 100 frames (starting at frame 100):

- grid resolution $10 \times 10 \times 10$ with $\delta\tau = 5.0$ and 2,344 liquid elements initially in 360 liquid cells with 1,000 total cells,
- grid resolution $20 \times 20 \times 20$ with $\delta\tau = 2.5$ and 16,362 liquid elements initially in 2,299 liquid cells with 8,000 total cells,
- grid resolution $30 \times 30 \times 30$ with $\delta\tau = 1.666$ and 52,900 liquid elements initially in 7,168 liquid cells with 27,000 total cells,
- grid resolution $40 \times 40 \times 40$ with $\delta\tau = 1.25$ and 122,758 liquid elements initially in 16,317 liquid cells with 64,000 total cells, and
- grid resolution $50 \times 50 \times 50$ with $\delta\tau = 1$ and 236,736 liquid elements initially in 31,096 liquid cells with 125,000 total cells.

Figure 5.10 shows several frames from the Scenario D simulation run with the 50×50 grid resolution.

Various data was collected from the different simulation runs of Scenarios C and D. The following sections present and discuss this data.

5.12.1 Rate of Convergence

For all simulation runs a conservative error tolerance $\epsilon = 0.0001$ was used. Figure 5.11 shows how many conjugate gradient iterations were required for the first frame in each simulation run of Scenario C that required nontrivial solution of the pressure Poisson equation. Figure 5.12 shows the rate of convergence for certain frames at the $20 \times 20 \times 20$ grid resolution. Figure 5.13 shows a detail of Figure 5.12.

Figure 5.14 shows the number of iterations required to reach the error tolerance for each frame in each simulation run of Scenario C. Figure 5.15 shows the resulting divergence in the liquid cells after the computed pressure is used to correct for the divergence.

Figure 5.16 shows the number of iterations required to reach the error tolerance for each frame in each simulation run of Scenario D. In these simulation runs, the solution of the pressure Poisson equation required on average about half of the total CPU time (the dominant stage of the simulation). Advecting the particles was the next most computationally intensive task. For reference, each frame of Scenario D required about five seconds of CPU time at the highest resolution ($50 \times 50 \times 50$) on a PC with a Pentium III 450 MHz CPU.

5.12.2 Variation in Element Motion with Grid Resolution

An interesting property of our simulation algorithm is how the computed motion of elements differs as the grid resolution is decreased (and the grid spacing is proportionately increased so that the size of the spatial domain remains the same). To explore this property, both Scenarios C and D included 45 liquid “tracer” elements whose motion was tracked over each simulation run. Figure 5.17 shows the distribution of these “tracers” for Scenario C (with all other liquid elements not shown for clarity). Using the motion of these tracers in the highest grid resolutions used for Scenario C (i.e., $40 \times 40 \times 40$) and Scenario D (i.e., $50 \times 50 \times 50$) as references, we computed the mean difference from their reference positions for corresponding frames at lower resolutions. Figures 5.18 and 5.19 shows the results for Scenarios C and D respectively. As expected, the difference increases with the progression of the simulation and with coarser grid resolutions.

5.12.3 Variation in the Number of Liquid Cells Over Time

We model fluid as being incompressible and thus the volume of liquid in our simulations should remain constant over time. In our discretization, a cell is entirely liquid, gas or solid. Thus, the volume of liquid in our simulation can be measured by the total number of liquid cells. Recall that a cell is classified as liquid if it is not solid and it contains any liquid element. This scheme inherently involves some variation in the total number of liquid cells. For example, imagine starting with one liquid cell containing two elements that after one simulation iteration advects one of the elements out of the cells: the result is two liquid cells and at least a temporary doubling of the liquid volume.

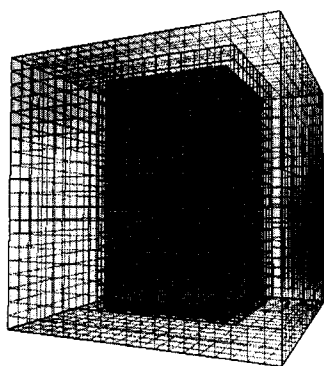
Figures 5.20 and 5.21 show the ratio of the total number of liquid cells to the

total number of grid cells for each simulation run of Scenario C and D respectively. The variation for these scenarios at these resolutions is quite small and is assumed to be visually insignificant though this possibility was not rigorously explored. The variation is clearly greatest for the smallest grid resolutions.

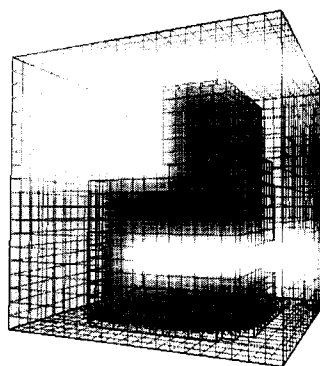
5.13 Discussion

The numerical techniques used in our simulation algorithm were motivated by a number of factors. Our approach to controlling flow simulation, discussed in Chapter 6, is based on the use of a Chorin projection scheme. Our approach to modelling multiple reactive fluids is based on the use of discrete elements to represent and track material properties. Thus, the Marker-and-Cell method for interface tracking is a natural approach. The need for efficient and stable simulation (even at the expense of accuracy) motivates the use of a fixed, uniform grid to discretely represent the spatial domain, the use of semi-Lagrangian integration for handling the convective terms, oversimplified interface conditions, and the modelling of the dynamics of inert flow for the animation of reactive flow. Naturally, the use of these numerical techniques has a number of important implications for computer animation, and so we consider them here.

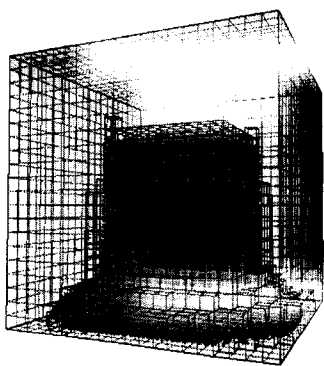
Perhaps most significantly, the CFL condition limits the maximum stable time step that can be taken for any given simulation iteration. This limitation directly corresponds to the maximal velocity in the simulation, and thus any violent behaviour anywhere in the domain will slow down the entire simulation. In particular, dramatic effects due to strong forces (or powerful control velocities from the streamtube flow primitives discussed in Chapter 6) are often desired in animated sequences, and thus this limitation can be particularly acute. Since we are simulat-



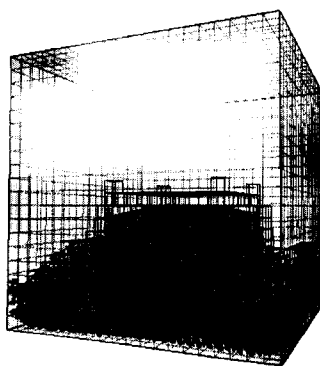
Frame 101



Frame 111

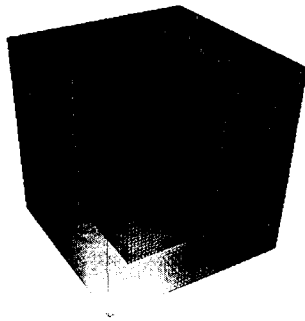


Frame 121

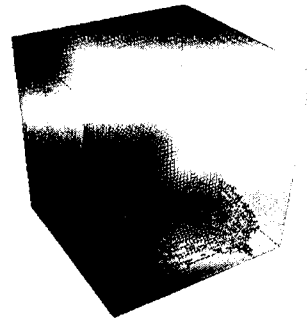


Frame 131

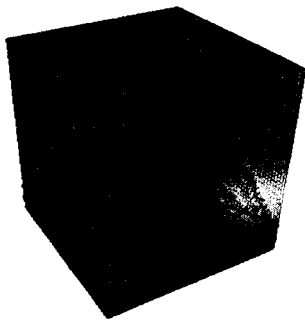
Figure 5.9: Frames from Scenario C with a $20 \times 20 \times 20$ grid resolution.



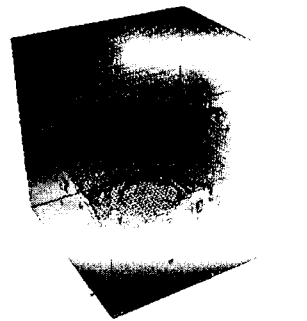
Frame 101



Frame 151



Frame 171



Frame 191

Figure 5.10: Frames from Scenario D with a $50 \times 50 \times 50$ grid resolution.

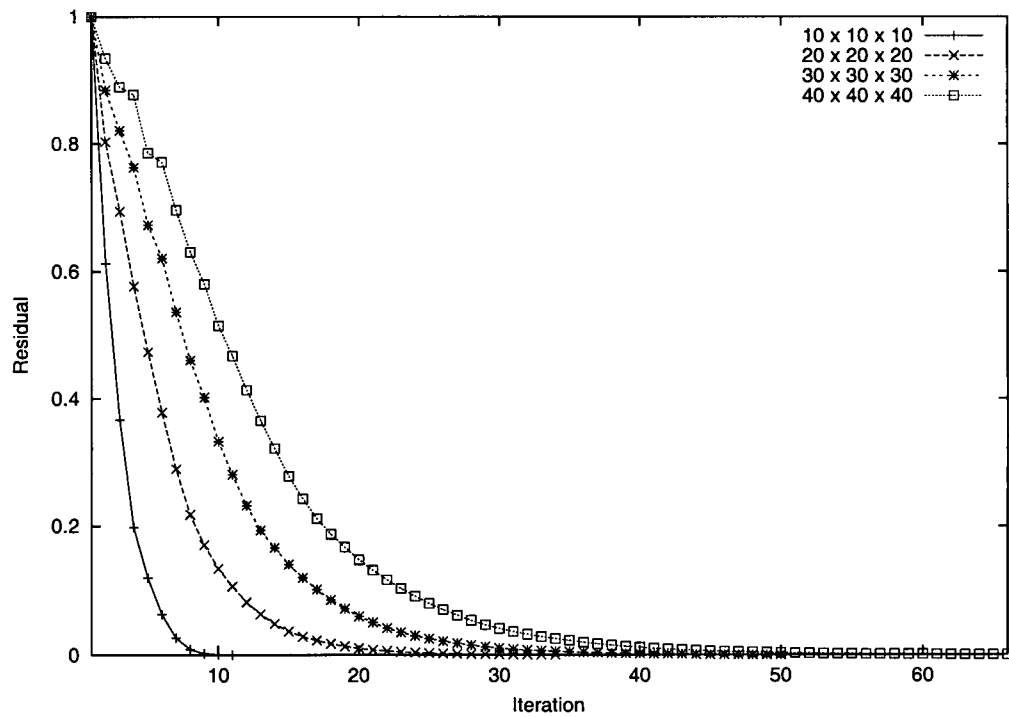


Figure 5.11: Rate of convergence for the solution of the pressure Poisson equation for the first frame in each simulation run of Scenario C requiring nontrivial solution (frame 100 for grid resolutions $10 \times 10 \times 10$ and $20 \times 20 \times 20$, frame 107 for grid resolution $30 \times 30 \times 30$, and frame 108 for grid resolution $40 \times 40 \times 40$).

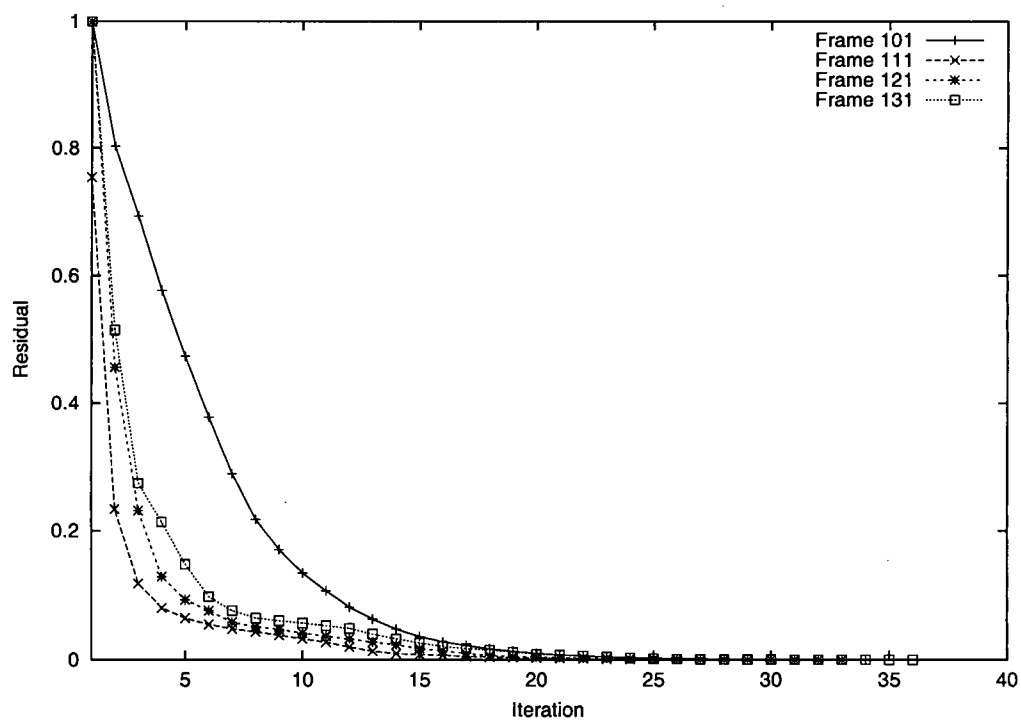


Figure 5.12: Convergence rates for the solution of the pressure Poisson equation for various frames of the simulation run of Scenario C with the $20 \times 20 \times 20$ grid resolution.

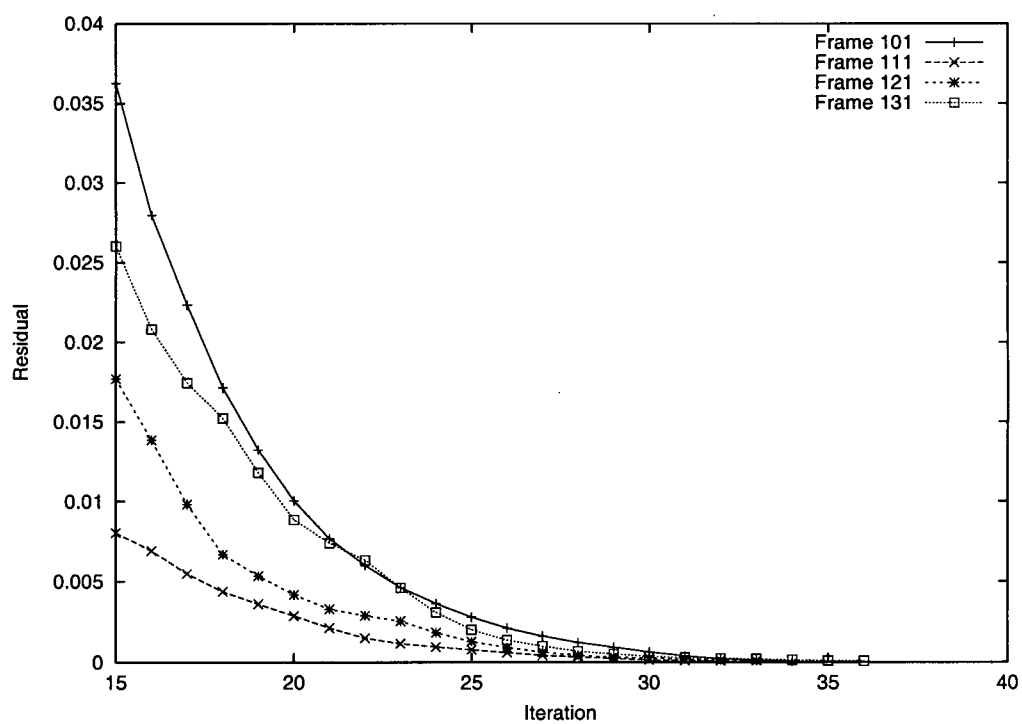


Figure 5.13: A detail of Figure 5.12.

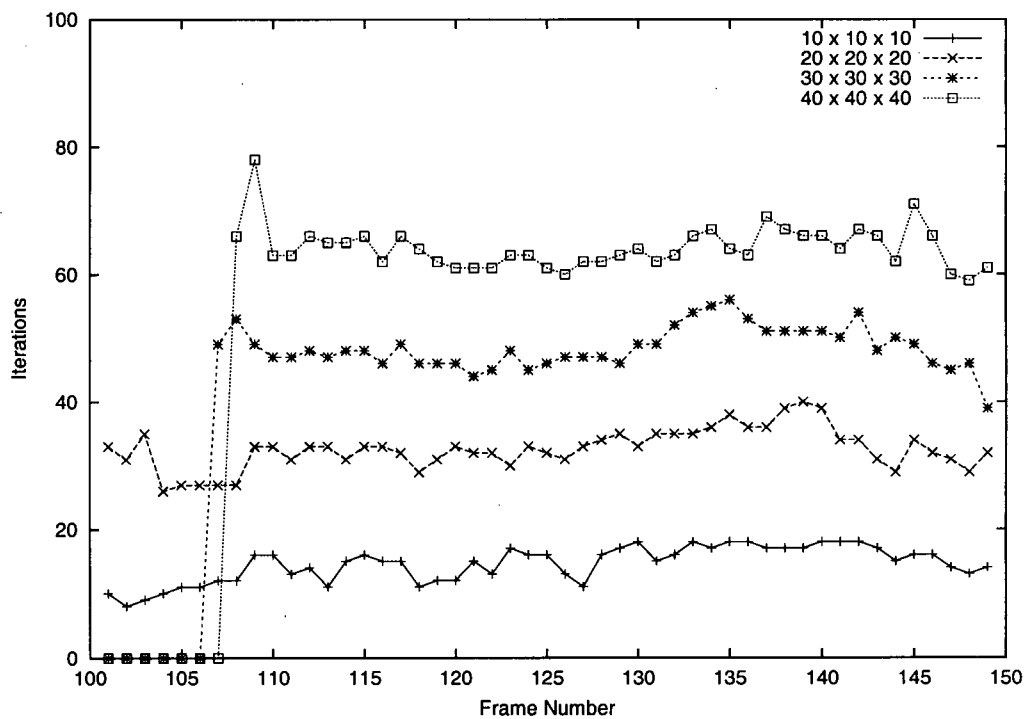


Figure 5.14: Number of iterations required to solve the pressure Poisson equation for Scenario C at various grid resolutions ($\epsilon = 0.0001$).

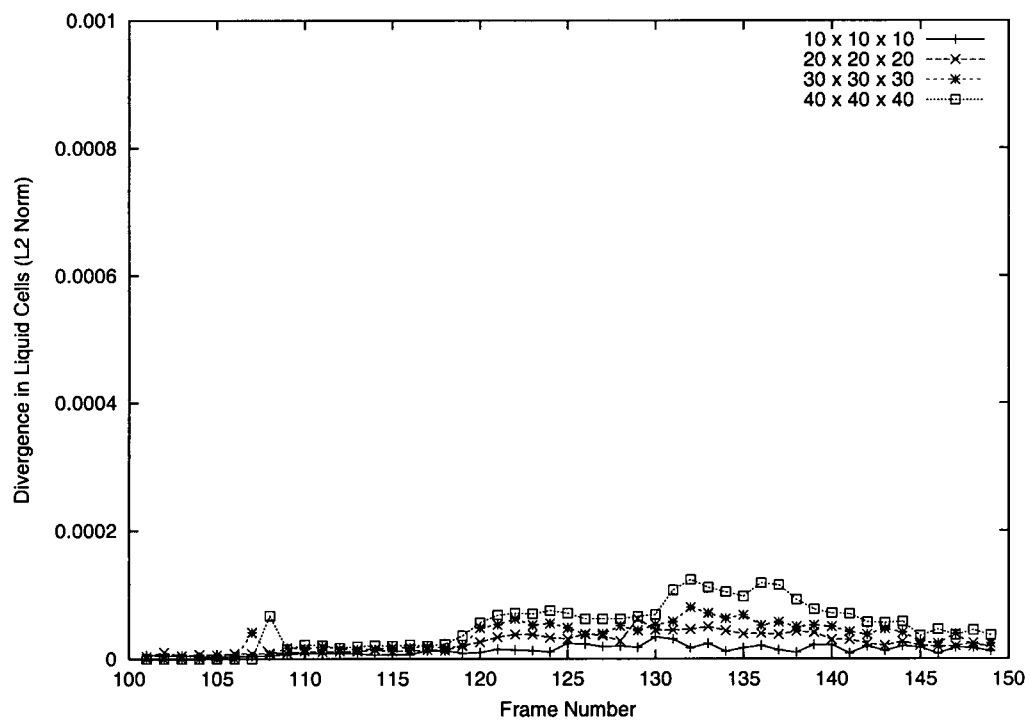


Figure 5.15: The divergence in liquid cells at each frame for various grid resolutions of Scenario C.

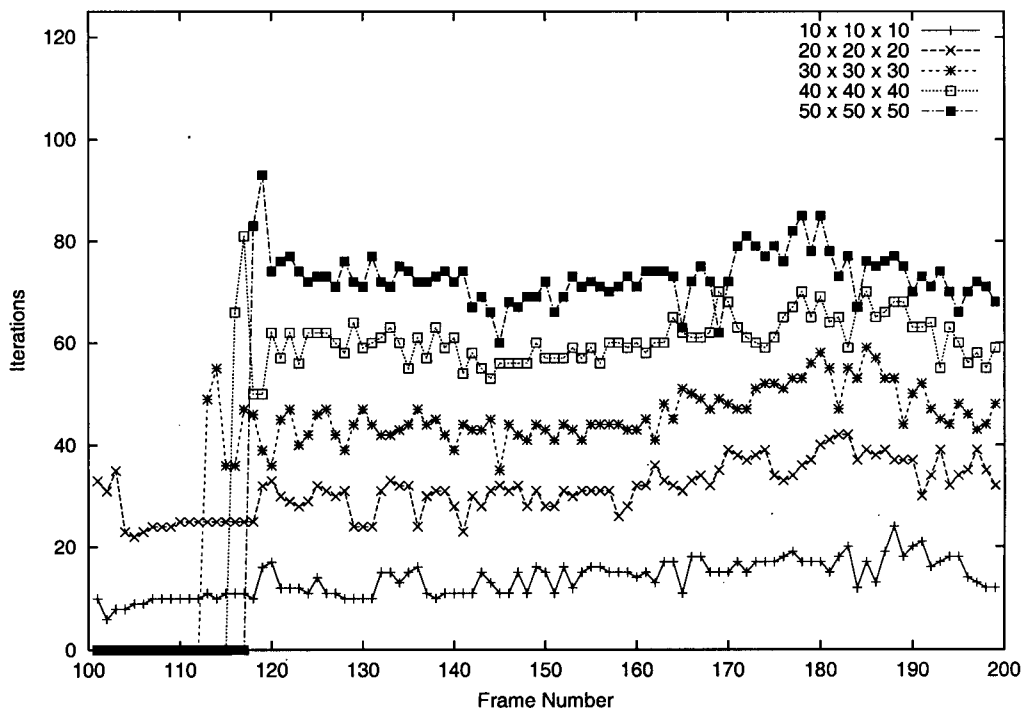


Figure 5.16: Number of iterations required to solve the pressure Poisson equation for Scenario D at various grid resolutions ($\epsilon = 0.0001$).

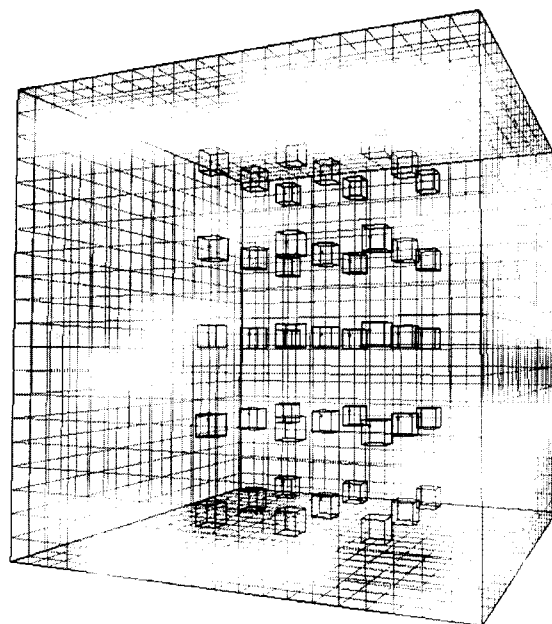


Figure 5.17: Tracer elements used in Scenario C.

ing inert flow, we sidestep the difficulties of simulating reactive flow. However, as discussed in Chapter 8, animators can program reactive elements to produce arbitrary amounts of heat resulting in strong thermal buoyancy forces that again slow down the simulation.

Because the solution of the pressure Poisson equation is often the critical step in the simulation algorithm, an important and related issue is the determination of an appropriate error tolerance for the iterative solution algorithm. We have used a conservative $\epsilon = 0.0001$. Determining the highest error tolerance that is acceptable for our visual simulation purposes is a difficult problem that we did not explore. In general this type of problem has not been adequately addressed in computer graphics.

The use a fixed uniform grid as our spatial discretization scheme imposes two

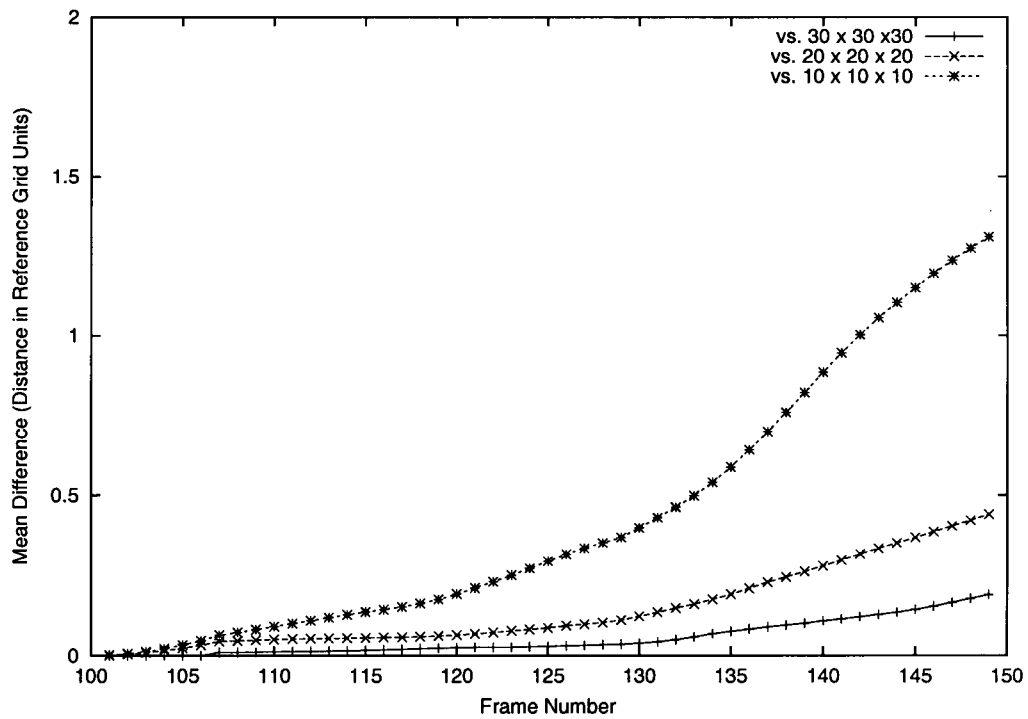


Figure 5.18: Mean difference in the positions of “tracer” elements with respect to their positions at each frame in the reference grid resolution ($40 \times 40 \times 40$) for Scenario C.

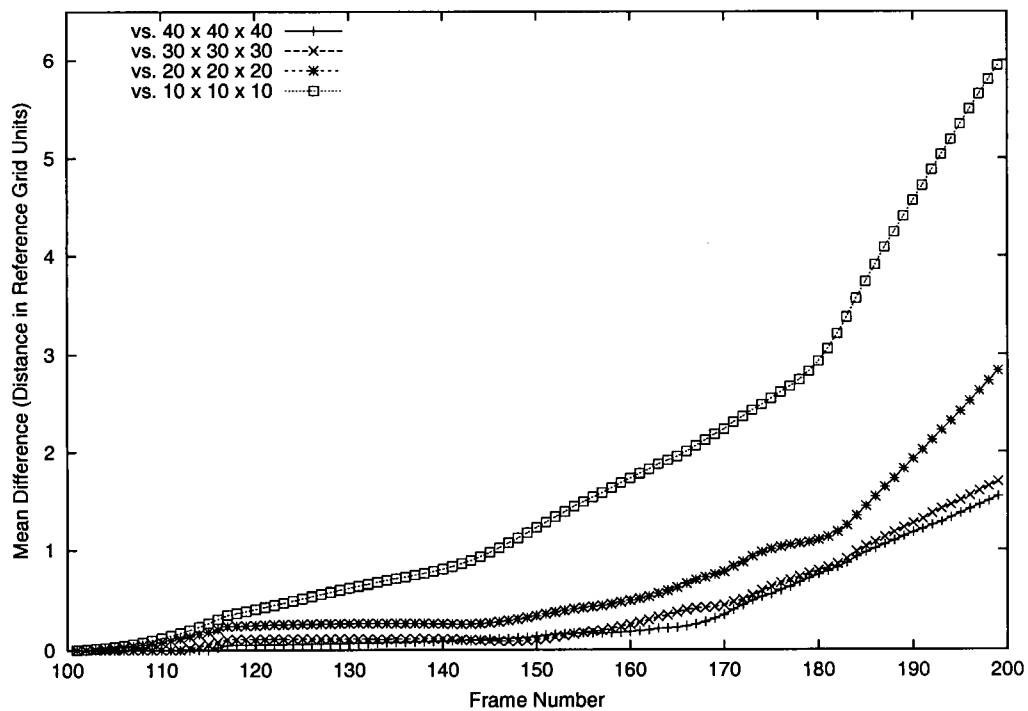


Figure 5.19: Mean difference in the positions of “tracer” elements with respect to their positions at each frame in the reference grid resolution ($50 \times 50 \times 50$) for Scenario D.

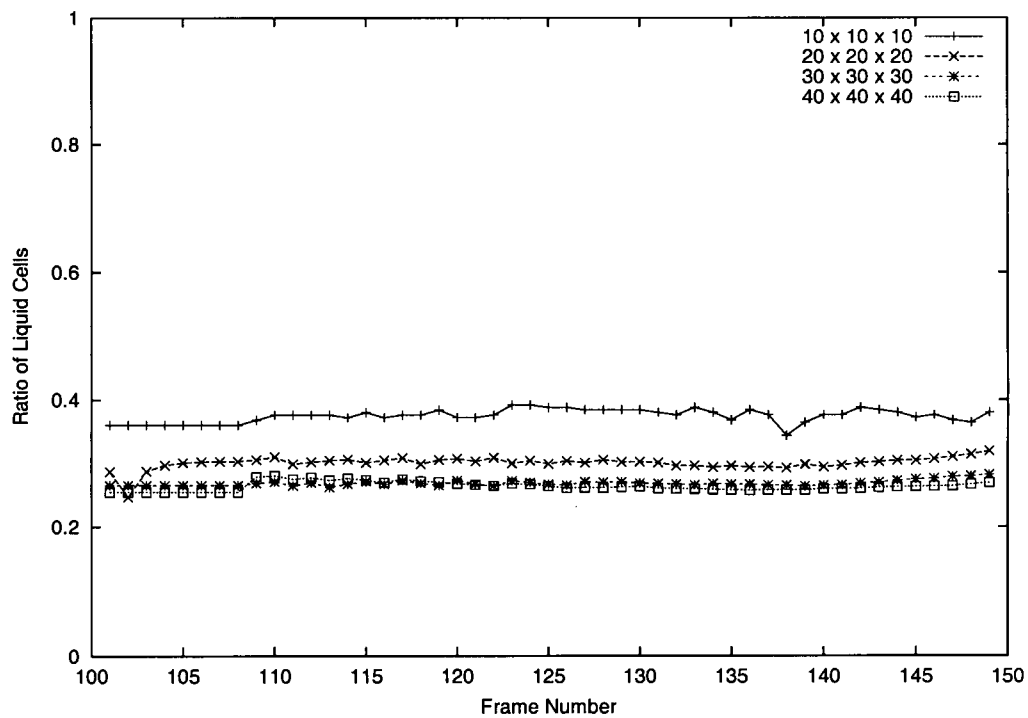


Figure 5.20: Ratio of the number of liquid cells to the total number of grid cells for each frame in Scenario C at various grid resolutions.

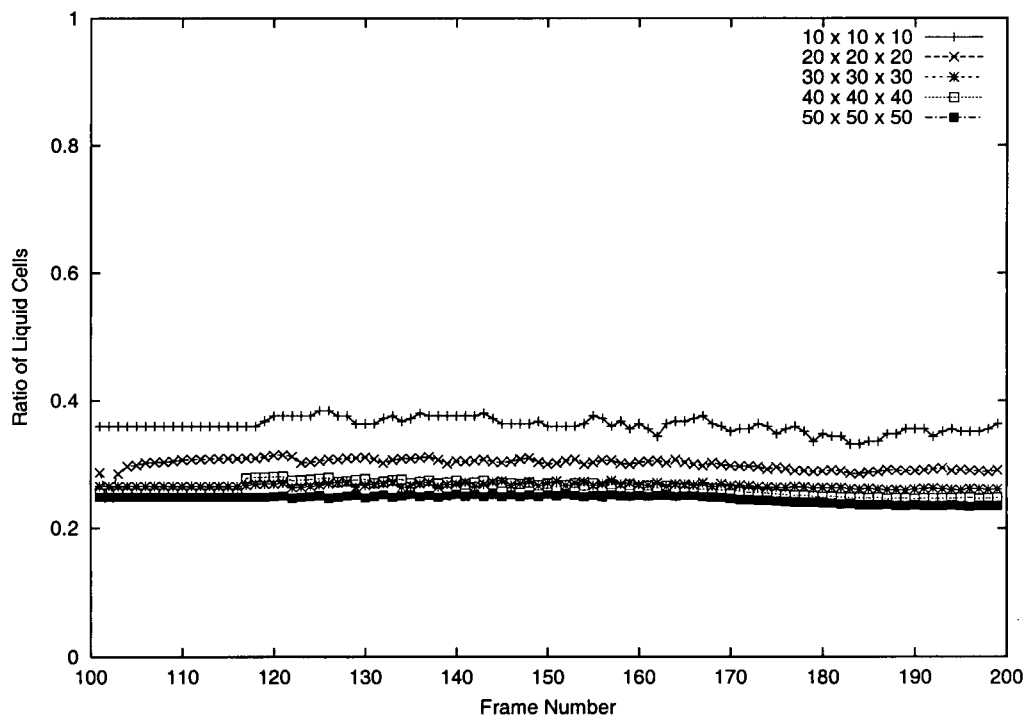


Figure 5.21: Ratio of the number of liquid cells to the total number of grid cells for each frame in Scenario D at various grid resolutions.

significant limitations. First, the range of scales that can be effectively modelled at the same time is obviously limited. Second, curved obstacles and boundaries must be represented (for simulation purposes only) as cubical cells. This second limitation may not be as severe as it appears since small inaccuracies (on a sub-cell scale) may be impossible to detect in rendered animations. Of course, the limitation implicit in axis-aligned boundaries still applies.

The use of marker elements requires elements to be distributed throughout the fluid domain to effectively represent and track multiple dynamic fluid materials. Enough elements need to be used to avoid spurious variations in liquid volume. It is not clear how to guarantee a sufficient number of elements, though in practice on the order of 50 elements per cell seems to work well. Also, over the evolution of a simulation run, marker elements tend to bunch into strands; we randomly perturb elements a small amount each iteration to avoid this, but there is no guarantee of avoiding pathological bunching behaviour. Finally, as discussed in Chapter 7, using discrete elements poses challenges for rendering a flat liquid surface.

Semi-Lagrangian integration is known to be overly diffusive. However, recent work [19] has presented methods for compensating for this drawback when the application is the computer animation of fluid.

Finally, our fluid model—while significantly more general than others used for computer animation—does not address many visually interesting fluid behaviours. In particular, we have oversimplified the gas-liquid interface conditions to facilitate efficient numerical simulation and have not addressed the dynamics of reactive flow. Thus, in our simulations, gas has no effect on the liquid (the pressure of the gas on the liquid is always taken to be zero). Gaseous bubbles, for example, simply collapse under liquid pressure.

Despite these limitations, our simulation algorithm is stable (subject to the CFL condition), and it is efficient for many animation scenarios. Its key advantage is that it addresses a much larger class of fluid phenomena than has been addressed before, and it is integrated with our control strategy, our shape and shading models, and our procedural animation method for chemical and thermal reactions.

Chapter 6

Controlling Flow Simulation

In this chapter we introduce a simple yet powerful technique of controlling incompressible flow simulation for computer animation purposes that works for any simulation method using a projection scheme for numerically solving the Navier-Stokes equations. In our technique, an abstract vector field representing the desired influence over the simulated flow is modelled using simple primitives. This technique allows an arbitrarily degree of control over the simulated flow at every point while still conserving mass, momentum, and energy.

6.1 Introduction

How can our flow simulation be “directed” to “perform” the specific, scripted behaviours often demanded in animation? For example, how could we simulate liquid flowing up out of one glass, performing a loop in midair, and flowing back down into another glass? Obviously appropriate initial and boundary conditions might be used, but this approach—if it works at all—can be unbelievably painstaking. Small changes in initial or boundary conditions can generate dramatically different results.

Boundary conditions can be adjusted over time, but their effects on the flow away from boundary may not be clear. Modelling external body forces can be a powerful approach, but it is often not clear to an animator what forces are required to overcome gravity, viscous forces, and the current fluid momentum in order to give a desired fluid velocity.

The challenge of controlling flow simulation for computer animation has received relatively little attention. Foster and Metaxas [25] presented several techniques for animator control over their liquid simulation method. In addition to setting initial and boundary conditions and modelling external body forces, these techniques include modifying pressure values to manipulate the simulated flow for effects such as the shock wave from an explosion. Foster and Fedkiw [22] allowed velocity values to be directly set anywhere in the flow by adapting their approach to modelling objects moving in fluids. They note their approach “[does not give] perfect direct control over the liquid motion.”

This thesis presents two basic strategies for simulation control. In this chapter we describe the first approach based on the interactive modelling of a vector field that can arbitrarily govern the simulated flow: from subtly influencing it at one location to directly controlling it at another. In Chapter 8 we discuss the second approach where reactive elements can be programmed to affect the flow.

6.2 The Influence Field

in-flu-ence [*from Latin influent-, influens, present participle of influere to flow in, from in- + fluere to flow – more at fluid*]

1(a): *an ethereal fluid held to flow from the stars and to affect the actions*

of humans...

—Merriam-Webster's Dictionary

A convenient physical metaphor for the will of the animator over simulated fluid flow is the specification of external body forces. The momentum equation (4.9) in our dynamic model includes a term for body forces, and thus we can model arbitrary changes in momentum. As mentioned previously, however, it is generally not clear what body forces are required to realized specific effects on the flow. Instead, our approach to controlling flow simulation is to directly model the effects of implied external body forces.

In our simulation algorithm, all momentum changes except for those due to the pressure gradient are computed to give an intermediate velocity field \mathbf{v}^* not constrained to conserve mass (cf. Section 5.6). At this point in the algorithm, we can simply set \mathbf{v}^* to whatever values we want by assuming there are corresponding external forces that effect these changes in momentum. The next stage of the simulation algorithm corrects the intermediate velocity so that the resulting flow conserves mass (has zero divergence).

Directly manipulating intermediate velocity values can be a powerful approach to controlling our flow simulation that is far more intuitive than modelling external forces. However, simply setting an intermediate velocity value at some grid point to an arbitrary value ignores the current momentum of the fluid as well as the effects of gravity and viscosity. For some scenarios, such a jarring change may be exactly what is desired, but in general a more subtle approach is desired: instead of clobbering the momentum we want to accelerate or decelerate the flow towards the desired behaviour, i.e., we want the implied body forces to be smoothly varying in space and time.

Another issue with manipulating the intermediate velocity \mathbf{v}^* is that the effects of the subsequent mass conservation step of the simulation algorithm on the velocity may be hard to predict. This step globally modifies the velocity field so that it has zero divergence. The more divergence we create anywhere in \mathbf{v}^* , the greater the global velocity correction will be, and in general the harder it becomes to predict its effects. Thus, we should try to not add any divergence if possible.

Our approach to controlling flow simulation is to define an *influence field* $\mathcal{I}(x, y, z, t)$:

$$\mathcal{I}(x, y, z, t) = \alpha_C(x, y, z, t)\mathbf{v}_C(x, y, z, t)$$

where

- $\mathbf{v}_C(x, y, z, t)$ is the *control velocity* and
- $\alpha_C(x, y, z, t) \in [0, 1]$ is the degree of control over simulated fluid velocity.

The control velocity $\mathbf{v}_C(x, y, z, t)$ is constrained to have zero divergence, i.e., to satisfy the continuity equation (4.2) for incompressible flow:

$$\nabla \cdot \mathbf{v}_C = 0 \tag{6.1}$$

We modify the computed intermediate velocity \mathbf{v}^* at time t_n as follows:

$$\mathbf{v}^*(\mathbf{x}) = (1 - \alpha_C(\mathbf{x}, t_n))\mathbf{v}^*(\mathbf{x}) + \alpha_C(\mathbf{x}, t_n)\mathbf{v}_C(\mathbf{x}, t_n) \tag{6.2}$$

so that the effect on the intermediate velocity varies depending on α_C , from unconstrained simulation ($\alpha_C = 0$) to explicit control ($\alpha_C = 1$). Thus, at any point where $\alpha_C > 0$ and $\mathbf{v}^* \neq \mathbf{v}_C$ there is an implied force driving the velocity towards \mathbf{v}_C . The magnitude of this implied force is proportional to α_C and $|\mathbf{v}^* - \mathbf{v}_C|$. This simple technique can be quite powerful—if there is an easy way to model the influence field.

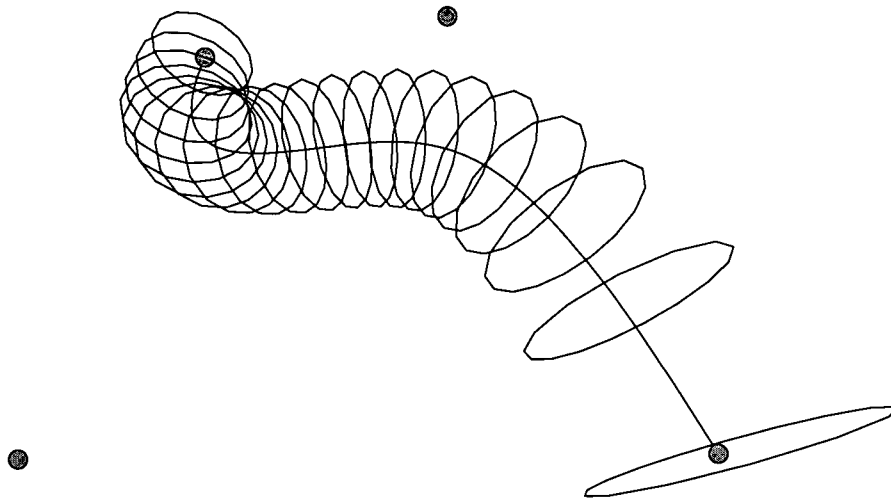


Figure 6.1: A simple streamtube flow primitive and its control points.

6.3 Streamtube Flow Primitives

To make our approach to controlling flow simulation useful, we need an effective method of interactively modelling the influence field $\mathcal{I}(x, y, z, t)$. The key challenge here is satisfying the mass-conservation constraint (6.1) on the control velocity \mathbf{v}_c for all points where $\alpha_c > 0$. In previous work [33], we addressed the challenge of modelling divergence-free flow fields (with infinite domains) using the superposition of divergence-free *flow primitives* \mathbf{v}_p , i.e.,

$$\nabla \cdot \mathbf{v}_p = 0,$$

and thus by the principle of superposition:

$$\nabla \cdot \left(\sum_p \mathbf{v}_p \right) = 0. \quad (6.3)$$

In other words, flow primitives are building blocks for constructing divergence-free flow fields.

We adapt this approach for modelling the influence field $\mathcal{I}(x, y, z, t) = \alpha_c \mathbf{v}_c$

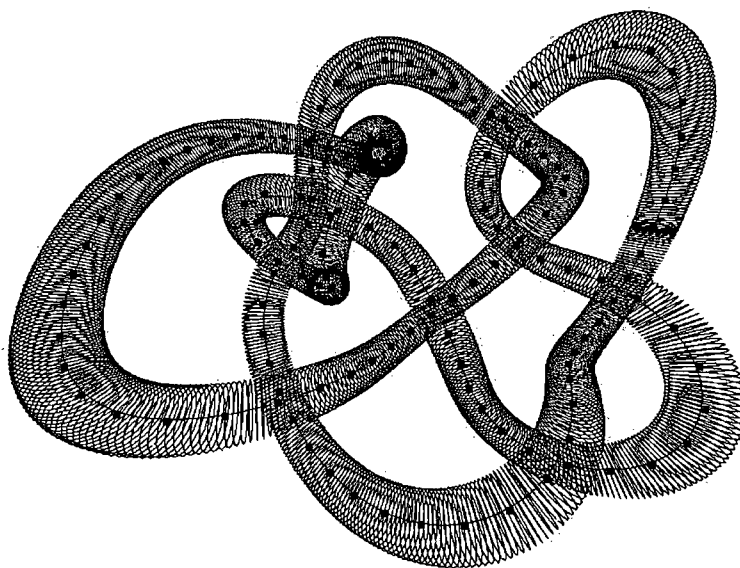


Figure 6.2: A more complex streamtube flow primitive. The control point data was generated using KnotPlot [65].

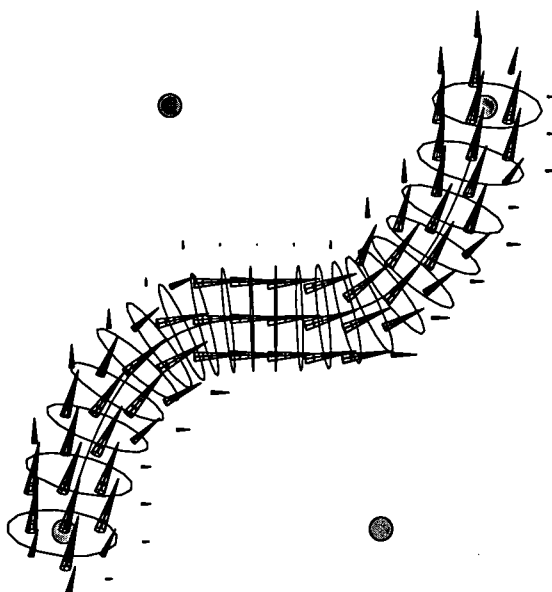


Figure 6.3: Flow in a streamtube primitive.

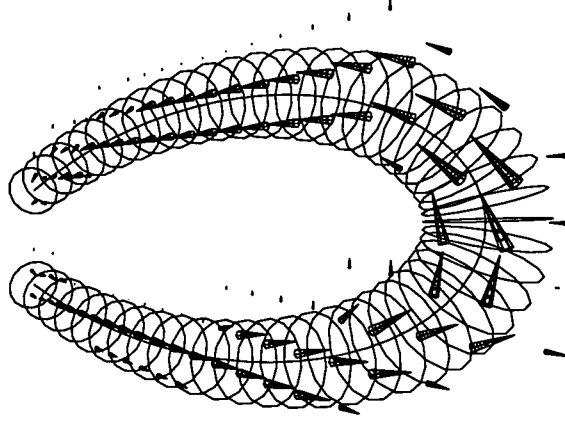


Figure 6.4: Flow in another streamtube primitive.

by using primitives $\mathcal{I}_p(x, y, z, t)$ where

$$\mathcal{I}_p(x, y, z, t) = \alpha_p(x, y, z, t) \mathbf{v}_p(x, y, z, t)$$

and

$$\alpha_c(x, y, z, t) = \min\left\{\sum_p \alpha_p(x, y, z, t), 1\right\} \quad (6.4)$$

$$\mathbf{v}_c(x, y, z, t) = \sum_p \mathbf{v}_p(x, y, z, t) \quad (6.5)$$

and thus

$$\mathcal{I} = \min\left\{\sum_p \alpha_p, 1\right\} \sum_p \mathbf{v}_p$$

Note that the mass-conservation constraint (6.1) on the control velocity is satisfied by equation (6.3).

An influence field primitive \mathcal{I}_p then is the product of a flow primitive \mathbf{v}_p and a scalar field primitive α_p . The influence field primitives are blended in a way such that resulting velocity control magnitude α_c is always in the range $[0, 1]$ and the resulting control velocity \mathbf{v}_c has zero divergence. We use α_p like a filter for flow

primitives \mathbf{v}_p where α_p is greatest in the region where the strongest influence is desired goes to zero moving away from that region. This facilitates blending as well as spatially smoothing out the influence of the primitive on the simulation.

The flow primitives \mathbf{v}_p in our previous work [33] had infinite domains; none of these primitives are suitable for our finite spatial domain Ω with its boundary constraint:

$$\int_{\Gamma} \mathbf{v} \cdot \mathbf{n} d\Gamma = 0$$

Thus, we have developed a new class of flow primitives based on streamtubes. A streamtube is a surface made up of streamlines starting at points forming a closed curve. There is no flow through this surface. In closed streamtubes (topologically equivalent to tori), the flow circulates entirely within the streamtube. In open streamtubes, the flux of fluid mass at one end of the tube equals the flux at the other end. As long as these streamtube flow primitives are completely within the fluid domain, they satisfy all the conditions of our dynamic problem.

Our streamtube flow primitives are naturally based on curves. Figure 6.1 shows a simple primitive based on a Bezier curve; Figure 6.2 shows a more complex one based on a cubic B-spline. Streamtube primitives can have variable thickness, i.e., the radial dimension can be a function of the curve parameter. We approximate the incompressible flow in a streamtube by numerically integrating a circular disc source along a curve. We do not require an *exact* solution here as our simulation algorithm will correct for any divergence in this numerical approximation. Figures 6.3 and 6.4 show the flow in two different streamtube primitives.

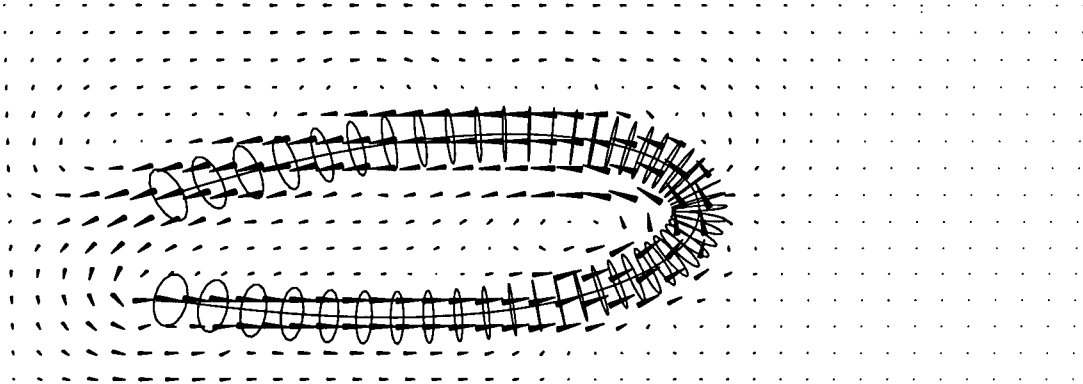


Figure 6.5: The streamtube primitive specifies the desired flow within the tube; the simulation must solve for the flow throughout the domain that matches the flow specified in the streamtube.

6.4 Results

In this section we present the results of applying our method of controlling flow simulation to a number of illustrative scenarios.

6.4.1 Basic Examples

First, we consider the simple case of a single streamtube flow primitive with its degree of control α_C set to unity throughout its domain ($\alpha_C \equiv 1$). This primitive is enclosed in a domain entirely filled with gas that is initially at rest. The domain has free-slip boundary conditions. After letting the simulation run for a short time, a steady state is achieved where, as shown in Figure 6.5, the flow follows the streamtube. Note that there is no discontinuity between the simulated region and the controlled region as the simulation algorithm solves for incompressible flow throughout the domain. Figure 6.6 shows a similar example with elements following the flow defined by a streamtube primitive. Figure 6.7 also shows a similar example, but with the streamtube crossed over itself. Note that the flow remains

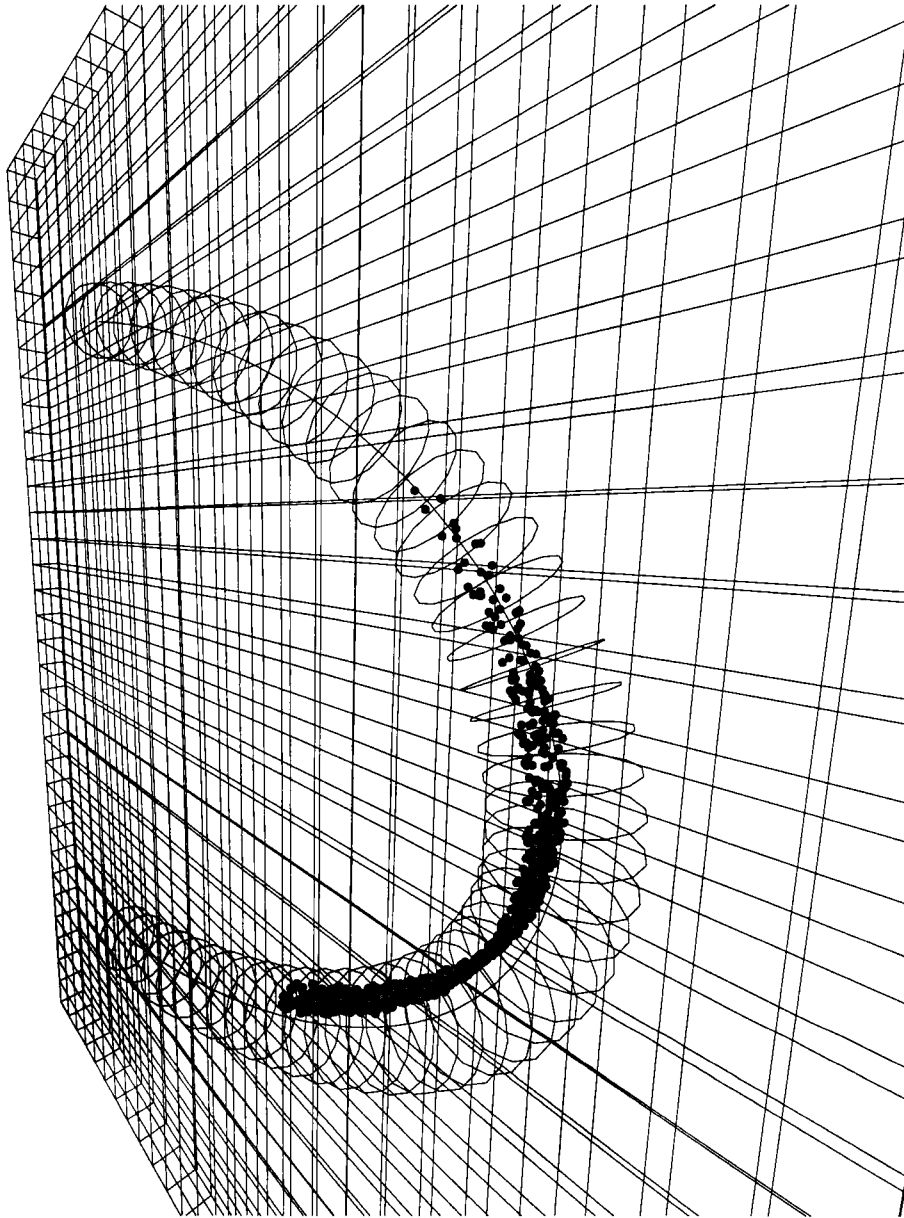


Figure 6.6: Elements following the desired flow specified by the streamtube primitives. Achieving this degree of control using forces would be quite difficult.

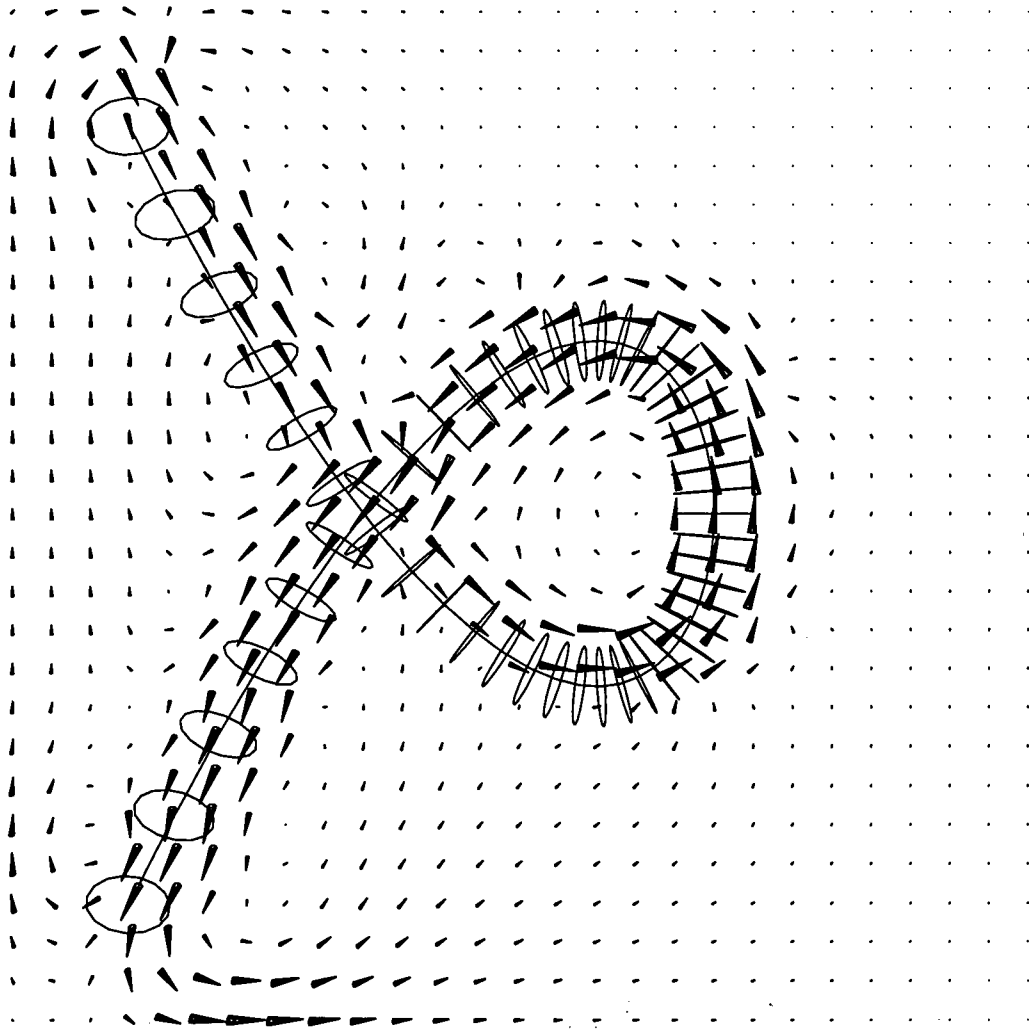


Figure 6.7: A streamtube primitive can cross itself; the simulation must still compute incompressible flow.

incompressible.

6.4.2 Varying the Degree of Control (α_C)

Next we consider the effect of varying the degree of control, α_C . In these examples, we use an orthographic view of a three-dimensional $40 \times 1 \times 40$ grid for clarity of illustration. Again, the domain is entirely filled with gas initially at rest and free-slip boundary conditions are used. The degree of control is represented by the opacity of the tube where $\alpha_C = 0$ is completely transparent and $\alpha_C = 1$ is completely opaque. Arrows along the defining curves of streamtube primitives show their flow direction, and arrows at the centers of computational cells show the computed flow. All simulations are run at 30 frames per second.

Figure 6.8 shows a closed streamtube flow primitive with $\alpha_C = 0.81$ at its center that radially decays to $\alpha_C = 0.04$ at its boundary at time $t = 0$. Figure 6.9 shows the flow after one simulation iteration (time $t = 0.03$), and Figure 6.10 shows the flow after ten iterations (time $t = 0.33$) when a steady state has been achieved.

Now consider the effect of adding a heat source at the center of the domain. First, we set the degree of control $\alpha_C = 0.08$. Figure 6.11 shows the flow at frame 127 where the flow due to thermal buoyancy caused by the heat source is visible but has not yet affected the flow influenced by the streamtube primitive. Figure 6.12 shows the flow at frame 157 where the interaction is apparent. Figure 6.13 shows the flow at frame 357 where the effect is pronounced, yet note even at $\alpha_C = 0.08$ there is an obvious influence on the simulated flow. Figures 6.14 and 6.15 show the same scenario with $\alpha_C = 0.43$. The influence is clearly stronger. Finally, we set $\alpha_C = 1.0$ at the boundary of the streamtube and $\alpha_C = 0.0$ at its center (in part to help visualize the simulated flow with the streamtube). As shown in Figure 6.16, at

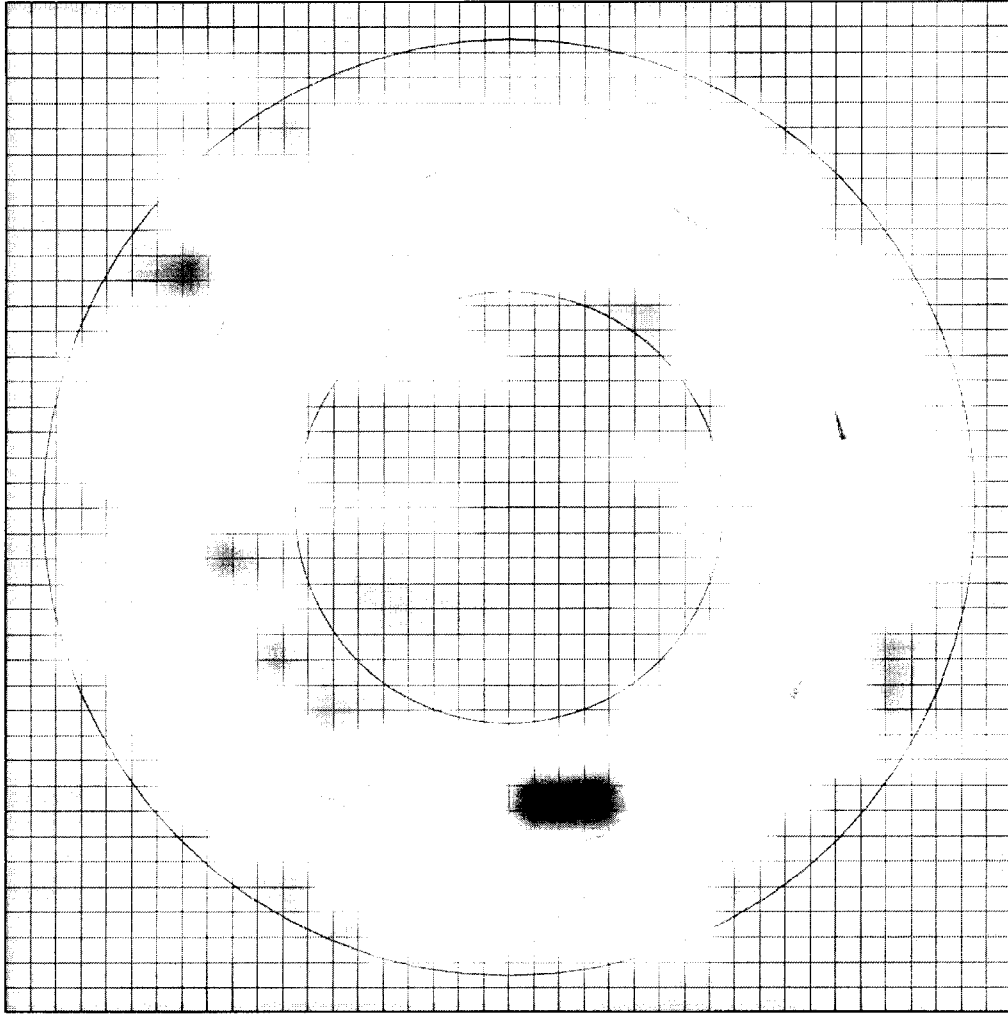


Figure 6.8: Orthographic view of a streamtube flow primitive with $\alpha_c = 0.81$ at its center decaying radially to $\alpha_c = 0.04$ at its boundary at time $t = 0$. The domain has free-slip boundary conditions and is filled with gas initially at rest. The opacity of the tube corresponds to the degree of control α_c where $\alpha_c = 0$ is completely transparent and $\alpha_c = 1$ is completely opaque.

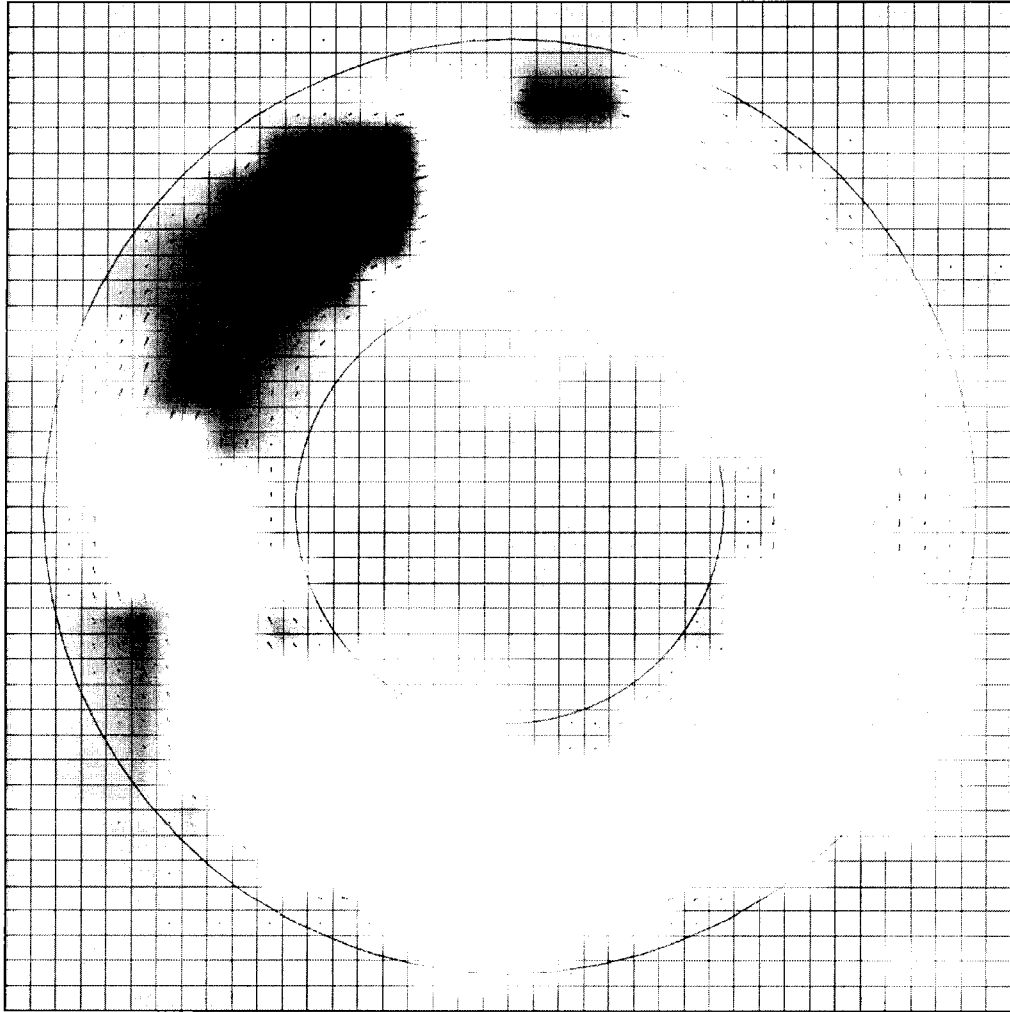


Figure 6.9: After one iteration ($t = 0.03$).

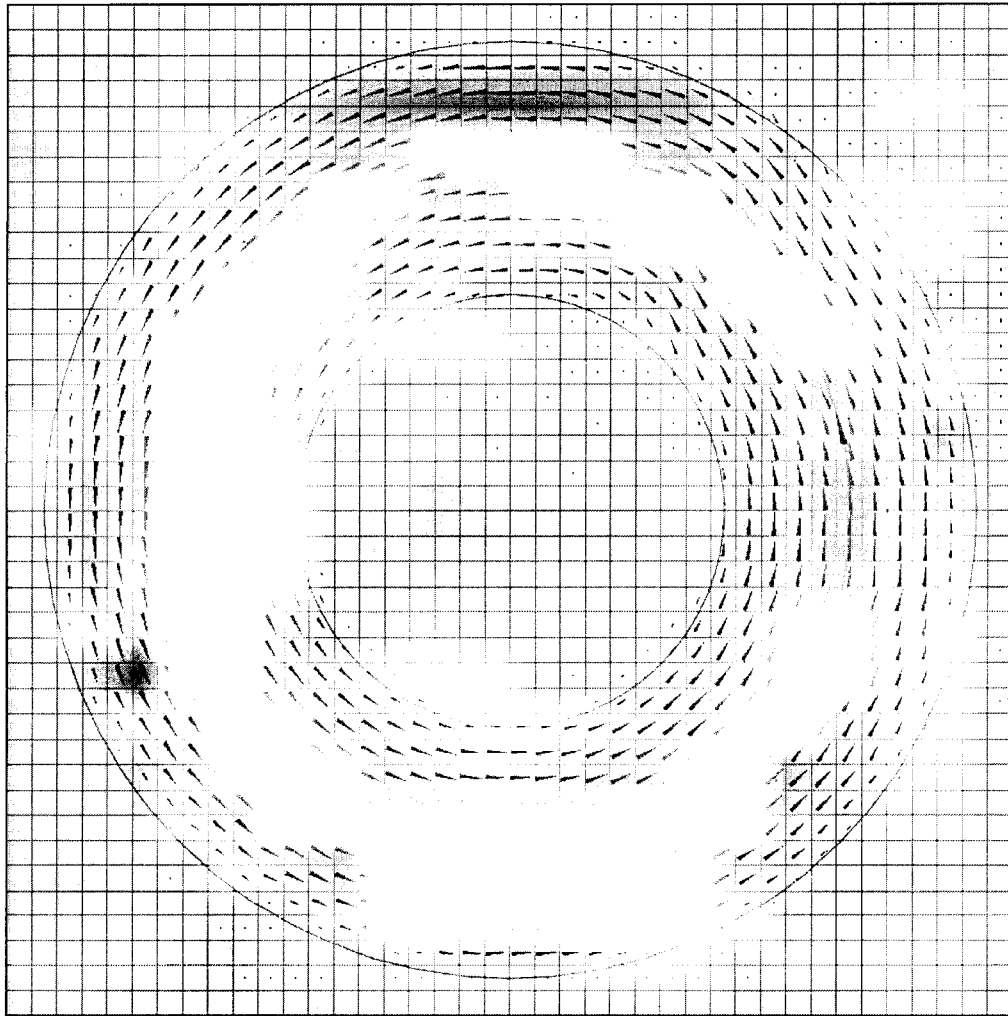


Figure 6.10: After ten iterations ($t = 0.33$).

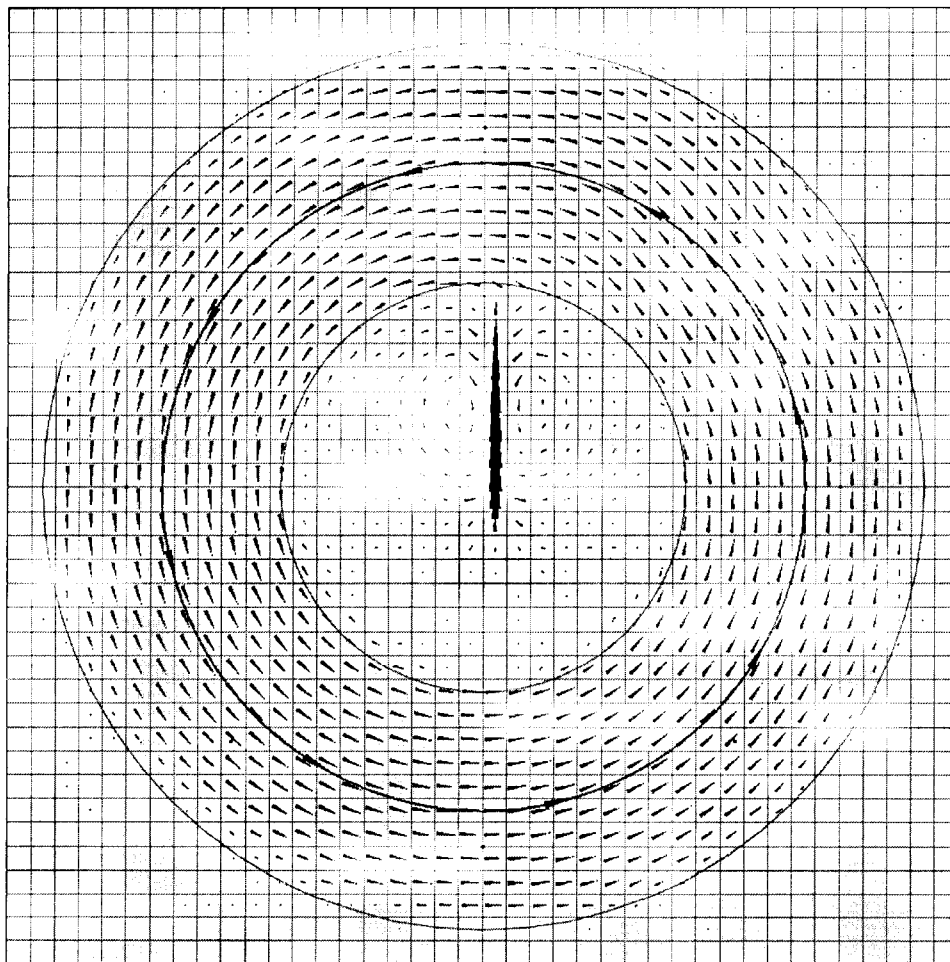


Figure 6.11: A streamtube flow primitive with $\alpha_C = 0.08$ surrounds a point heat source at the center of the domain (time $t = 0.9$).

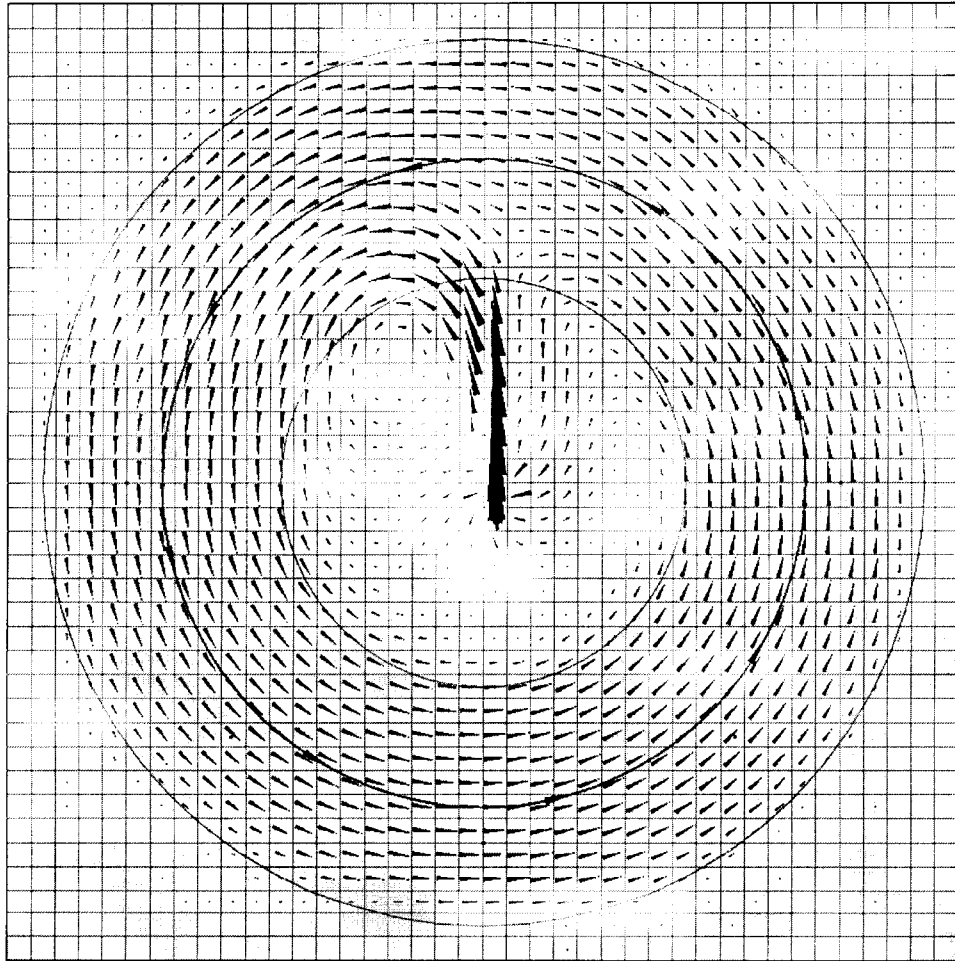


Figure 6.12: A streamtube flow primitive with $\alpha_c = 0.08$ surrounds a point heat source at the center of the domain (time $t = 1.9$).

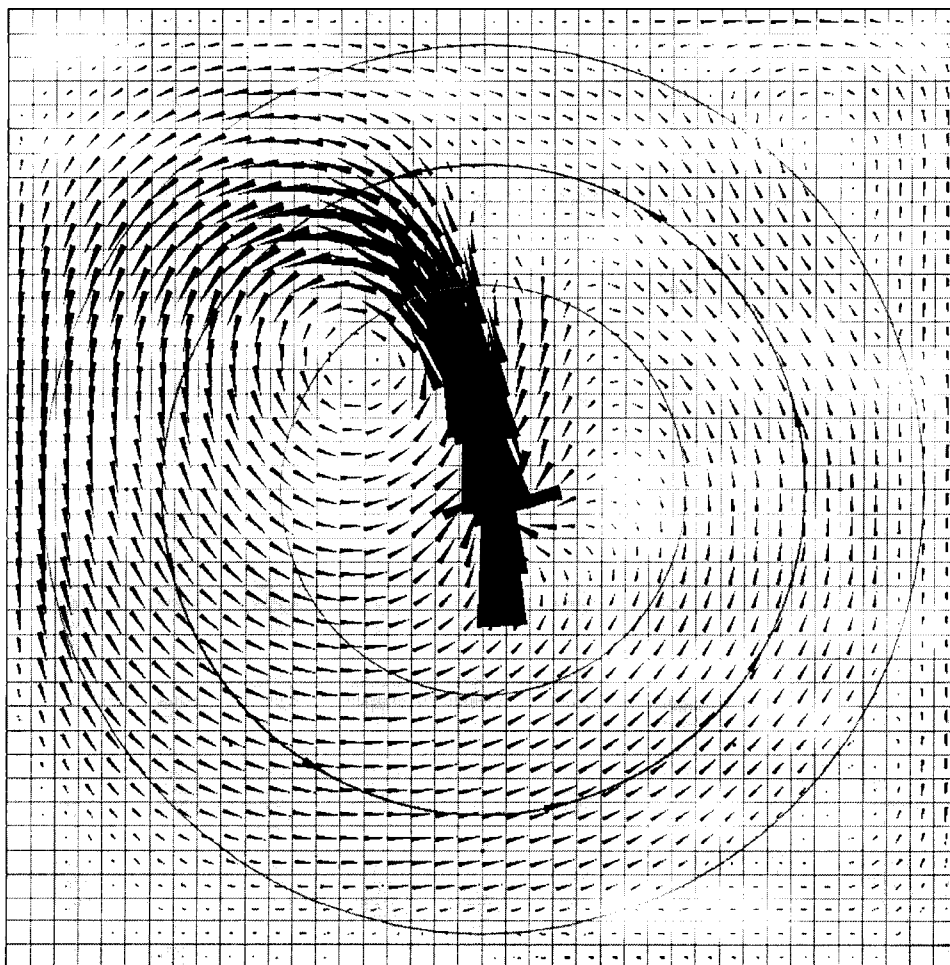


Figure 6.13: A streamtube flow primitive with $\alpha_C = 0.08$ surrounds a point heat source at the center of the domain (time $t = 8.57$).

this degree of control, the streamtube acts like a virtual boundary.

6.4.3 Composition of Streamtube Flow Primitives

So far we have considered a single streamtube primitive and the effects of varying its degree of control. Now we consider the composition of multiple streamtube primitives. Figure 6.17 shows two streamtube flow primitives with constant $\alpha_C = 0.28$ that overlap in some regions after a steady state has been reached. In these regions, the α_C value is defined to be the sum of the α_C values of each streamtube (clamped to a maximum of $\alpha_C = 1$). An alternate approach for overlapping regions would be to use the maximal α_C value of all streamtubes that influence a point in the domain. However, we argue that it is more intuitive to sum the values.

Figure 6.18 shows the same two streamtube flow primitives but with $\alpha_C = 0.5$ at the center of the streamtube radially decaying to zero. Figure 6.19 shows the flow after a steady state has been reached. Note the differences with Figure 6.17.

While it may be more intuitive to compose streamtubes flow primitives in a complementary fashion where the flow directions are not opposed, it is of course possible to overlap streamtube flow primitives that oppose each other in flow direction as shown in Figure 6.20.

6.4.4 Goal-Directed Fluid Flow

Finally, we apply our method of controlling flow simulation to a specific animation task. This task consists of modelling a clear fluid with text (the word “control”) written in dye that flows in a manner such that the text follows a particular path. We represent the text using an image that is texture-mapped via texture coordinates that are advected with the simulated flow. Initially these texture coordinates are

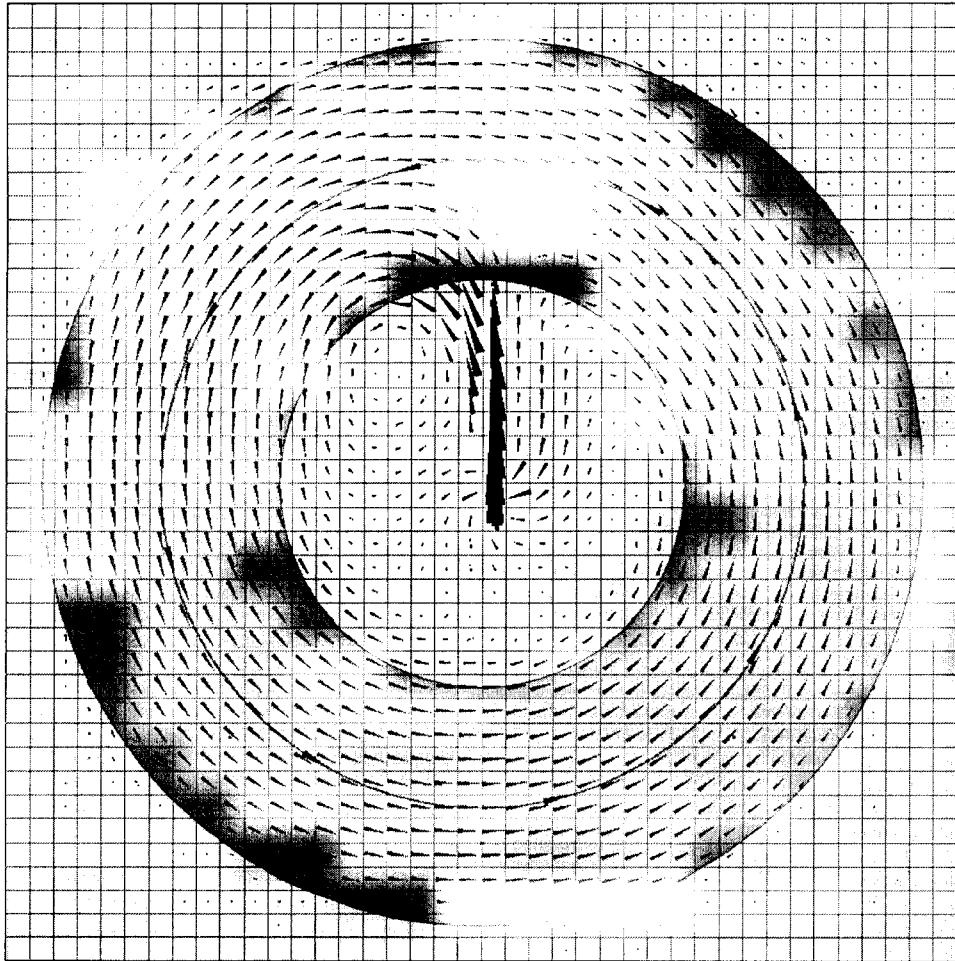


Figure 6.14: A streamtube flow primitive with $\alpha_c = 0.43$ surrounds a point heat source at the center of the domain (time $t = 2.13$).

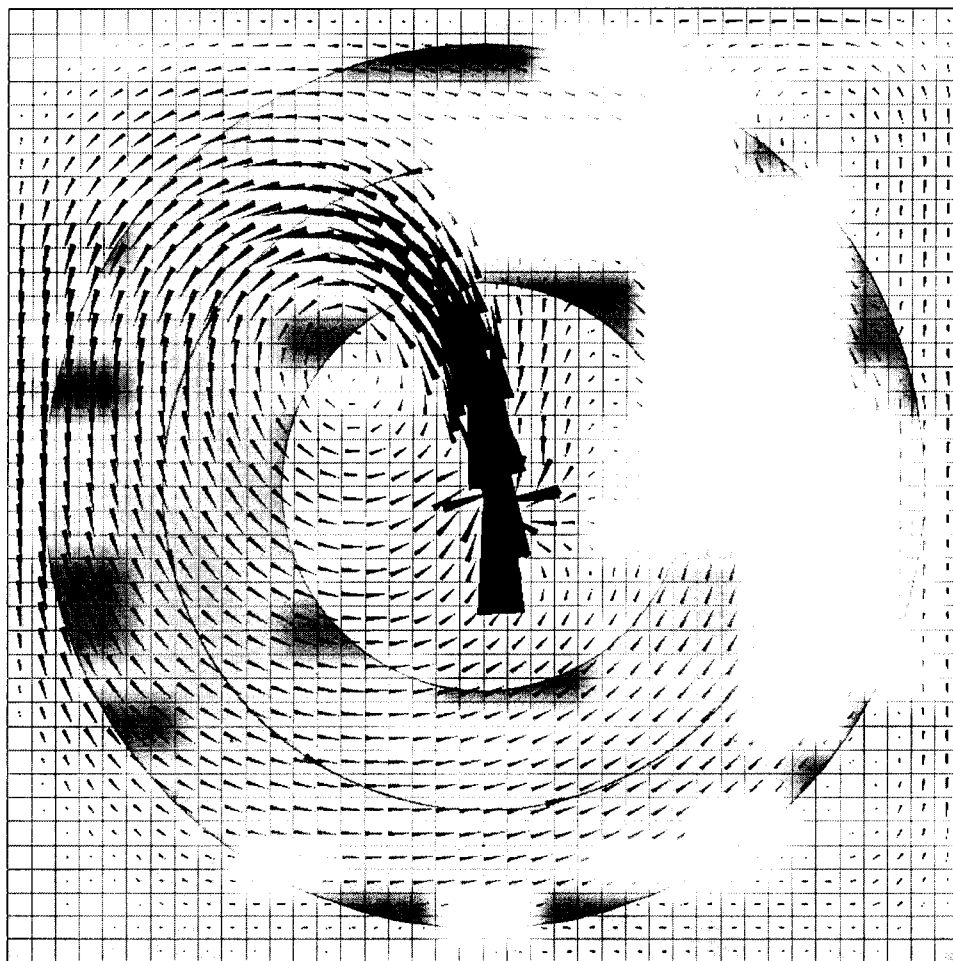


Figure 6.15: A streamtube flow primitive with $\alpha_C = 0.43$ surrounds a point heat source at the center of the domain (time $t = 8.4$).

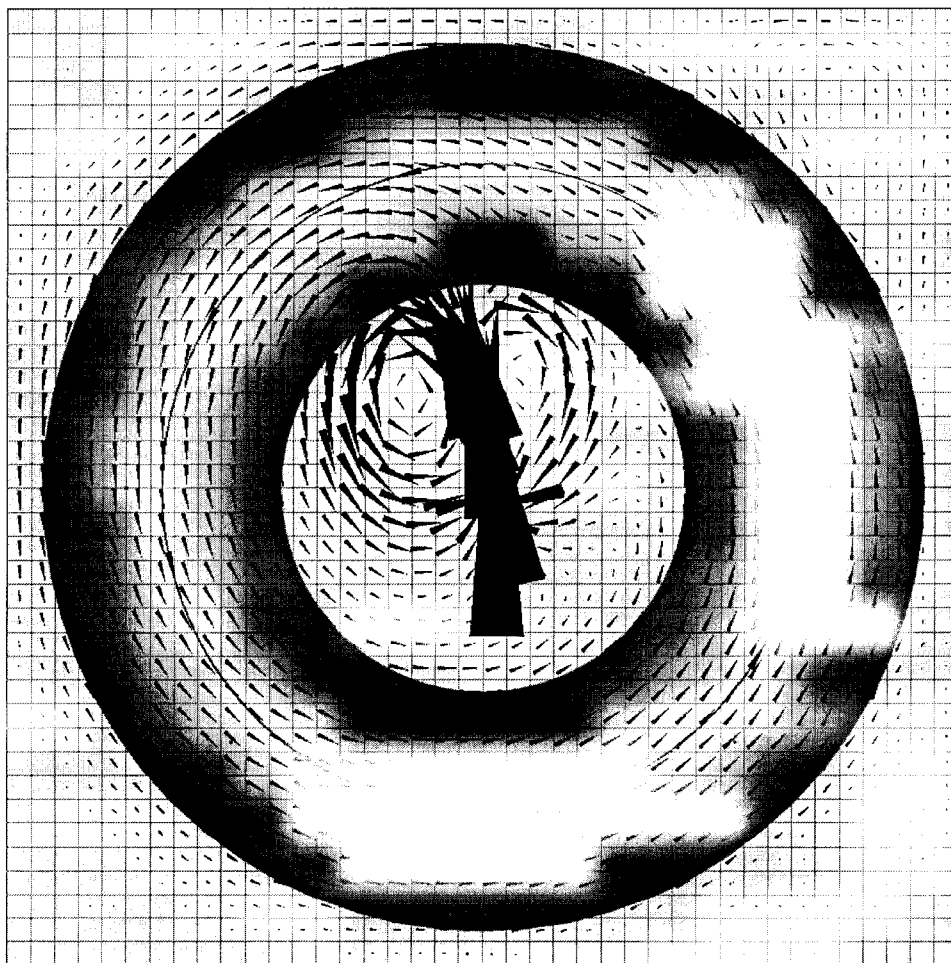


Figure 6.16: A streamtube flow primitive with $\alpha_C = 1$ at its boundary and $\alpha_C = 0$ at its center surrounds a point heat source at the center of the domain (time $t = 8.26$).

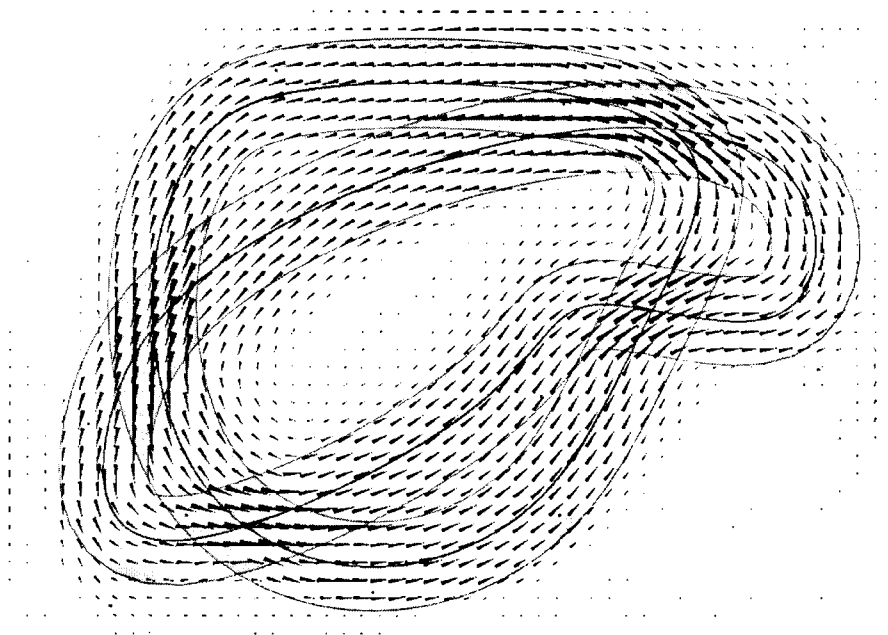


Figure 6.17: Composition of two streamtubes with constant $\alpha_c = 0.28$.

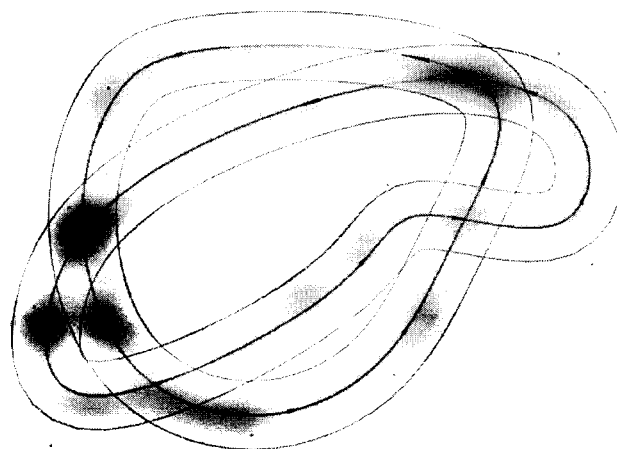


Figure 6.18: Composition of two streamtubes with radially decaying α_C , from $\alpha_C = 0.5$ to $\alpha_C = 0.0$, at time $t = 0.0$.

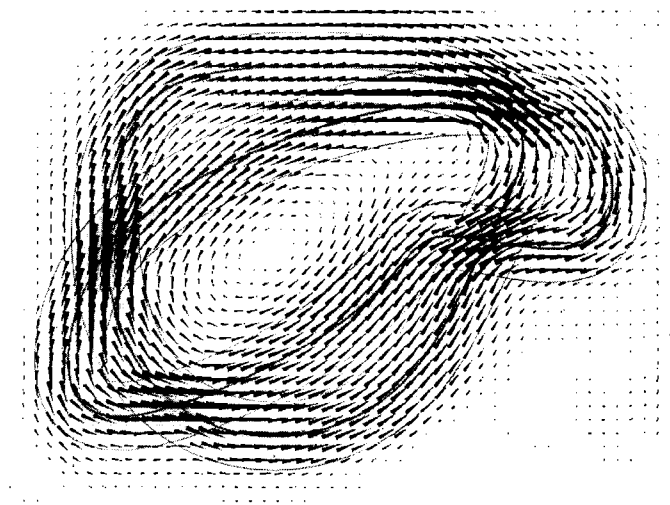


Figure 6.19: Composition of two streamtubes with radially decaying α_C , from $\alpha_C = 0.5$ to $\alpha_C = 0.0$, at time $t = 14.13$.

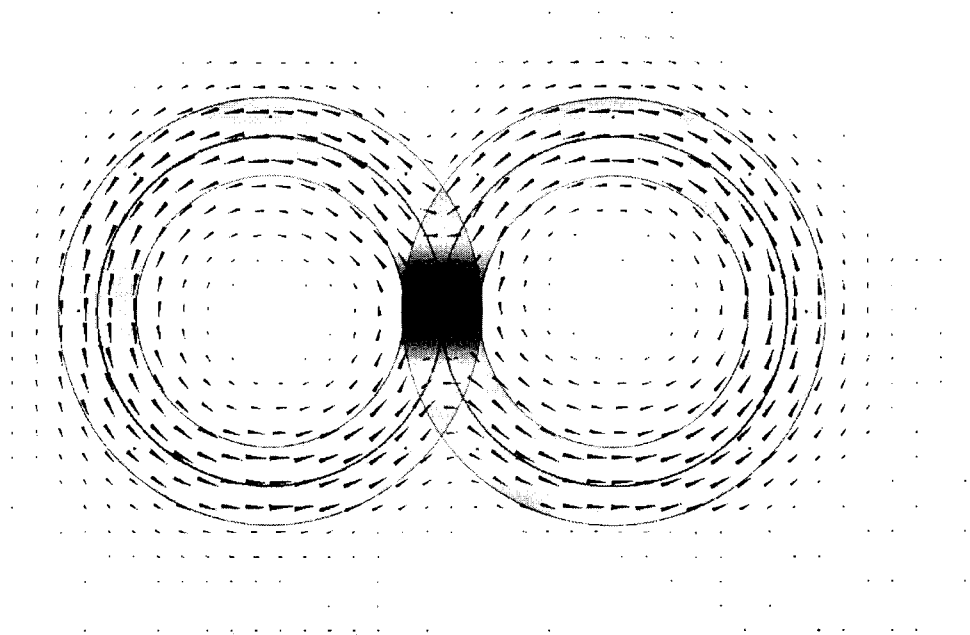


Figure 6.20: Two streamtube flow primitives with the same flow magnitude that have opposing flow directions in the region where they overlap. Both primitives have $\alpha_C = 0.3$.

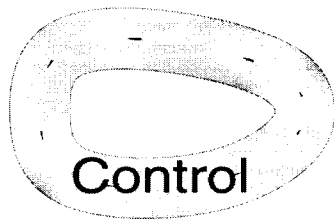
scalar fields ramped from zero to one in each coordinate direction.

This goal is easily realized with a streamtube flow primitive (with $\alpha_C = 0.23$). Figure 6.21 shows six frames from the resulting animation sequence. Note that there is no dependency of the texture on the streamtube; the texture simply moves with the simulated flow which in turn is influenced by the streamtube.

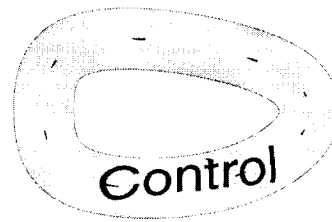
6.5 Discussion

We have shown that streamtube flow primitives can be a powerful approach to controlling incompressible flow simulation based on a Chorin projection method. The potential of this approach, however, has not been fully explored. For example, we have not investigated the animation of streamtubes. All parameters of a streamtube flow primitive can be animated: the control points, the flow magnitude, the radius, and the degree of control. We conjecture that the translation of a streamtube flow primitive with flow circulating radially around the defining curve (as opposed to tangentially with it) could be an effective means of generating waves.

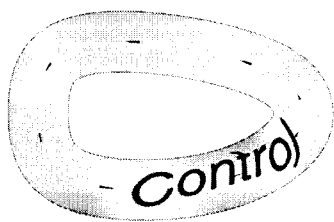
Naturally our approach to controlling fluid flow simulation for computer animation purposes has a number of basic limitations. As presented, streamtube flow primitives must be entirely within fluid cells and cannot cross fluid-solid interfaces or domain boundaries. (If they do, the control velocity may not satisfy the continuity equation.) Also, while our approach greatly improves the predictability of how a flow simulation will progress, it remains a trial-and-error approach without any guarantees of how the simulated flow will behave.



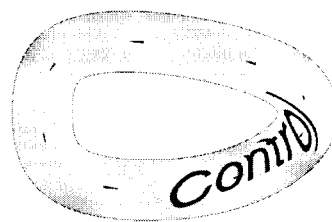
$t = 0$



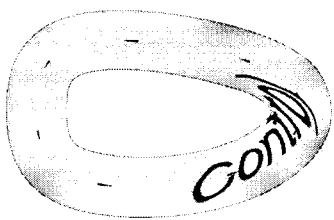
$t = 0.56$



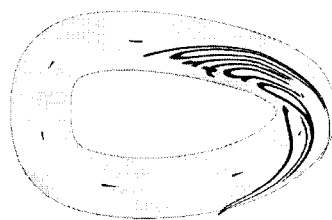
$t = 1.23$



$t = 1.8$



$t = 2.3$



$t = 4.06$

Figure 6.21: An example of goal-directed flow simulation.

Chapter 7

Shape and Shading

In this chapter we describe how we synthesize realistic images from our fluid model. We use implicit surface functions for liquid surfaces and volume density functions for participating media such as steam, smoke, and dust. We show how these shape models can be shaded using material properties and other parameters of our integrated fluid model. We also discuss our implementation of a bidirectional Monte Carlo ray-tracer for rendering realistic images from these shape and shading models.

7.1 Basic Principles of Image Synthesis

Before describing our shape and shading models and how we render images from them, we first review the basic principles of *image synthesis* or *rendering*, i.e., the process of computing how light is emitted, propagated, scattered, and absorbed throughout a scene to arrive at a given viewpoint. Our scenes consist of surfaces and participating media, and thus we review surface scattering and light transport in participating media. We also review the *radiance equation* which incorporates these processes and summarizes the rendering problem.

7.1.1 Surface Scattering

The scattering of light at a surface is given by the *bidirectional scattering distribution function* (BSDF) $f_S(\vec{\omega}', \vec{\omega})$ which gives the radiance leaving in direction $\vec{\omega}$ per unit of irradiance arriving from $\vec{\omega}'$. The radiance $L(\mathbf{x}, \vec{\omega})$ leaving a surface point \mathbf{x} in direction $\vec{\omega}$ is given by the surface scattering equation:

$$L(\vec{\omega}) = \int_{\Theta} L_i(\vec{\omega}') f_S(\vec{\omega}', \vec{\omega}) d\vec{\omega}' \quad (7.1)$$

which integrates the product of the BSDF and the incident illumination L_i over the sphere Θ of all directions. Equation (7.1) can be used to compute the appearance of points on a surface from a given viewpoint.

A *shading model* for a surface is generally an approximation of the BSDF. Typically, the BSDF is decomposed into the bidirectional reflection distribution function (BRDF) f_r and the bidirectional transmission distribution function (BTDF) f_t which describe the scattering for their respective hemispheres.

7.1.2 Light Transport in Participating Media

A participating medium can emit, absorb, and scatter light. To describe these processes it is necessary to define some terms:

- the *emission* $L_e(\mathbf{x}, \vec{\omega})$ is the radiance emitted at the point \mathbf{x} in direction $\vec{\omega}$,
- the *absorption coefficient* $\sigma_a(\mathbf{x})$ is the fraction of radiance lost per unit distance to other energy forms such as heat,
- the *scattering coefficient* $\sigma_s(\mathbf{x})$ is the fraction of the incoming radiation scattered from its original direction,

- the *extinction coefficient* σ_t is the sum of the absorption and scattering coefficients, $\sigma_t = \sigma_a + \sigma_s$, where the *mean free path* that a photon can travel without being scattered or absorbed is $1/\sigma_t$,
- the *scattering albedo* is the ratio of the scattering coefficient to the extinction coefficient, σ_s/σ_t , where low albedo values (near zero) correspond to dark media and high albedo values (near one) correspond to bright media, and
- the *phase function*, $p(\mathbf{x}, \vec{\omega}', \vec{\omega})$ is a scattering function giving the fraction of radiance from direction $\vec{\omega}'$ being scattered in direction $\vec{\omega}$.

For light interacting with a participating medium, the change in radiance at a point \mathbf{x} in direction $\vec{\omega}$ is given by the integro-differential transport equation:

$$\frac{dL(\mathbf{x}, \vec{\omega})}{d\mathbf{x}} = \sigma_a L_e - \sigma_t L(\mathbf{x}, \vec{\omega}) + \sigma_s \int p(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(\mathbf{x}, \vec{\omega}') d\vec{\omega}' \quad (7.2)$$

Integration of equation (7.2) along a straight path from \mathbf{x}_0 to \mathbf{x} in direction $\vec{\omega}$ gives:

$$\begin{aligned} L(\mathbf{x}, \vec{\omega}) &= \int_{\mathbf{x}_0}^{\mathbf{x}} \tau(\mathbf{x}', \mathbf{x}) \sigma_a(\mathbf{x}') L_e(\mathbf{x}', \vec{\omega}) d\mathbf{x}' \\ &+ \tau(\mathbf{x}_0, \mathbf{x}) L(\mathbf{x}_0, \vec{\omega}) \\ &+ \int_{\mathbf{x}_0}^{\mathbf{x}} \tau(\mathbf{x}', \mathbf{x}) \sigma_s(\mathbf{x}') \int_{\Theta} p(\mathbf{x}', \vec{\omega}', \vec{\omega}) L(\mathbf{x}', \vec{\omega}') d\vec{\omega}' d\mathbf{x}' \end{aligned} \quad (7.3)$$

where $\tau(\mathbf{x}', \mathbf{x})$ is the *transmittance* along the line segment from \mathbf{x}' to \mathbf{x} :

$$\tau(\mathbf{x}', \mathbf{x}) = e^{-\int_{\mathbf{x}'}^{\mathbf{x}} \sigma_t(\xi) d\xi}$$

The fact that the radiance at a point in the scene directly depends on the radiance of all other points in the visible space around it makes the solution for scenes with participating media much more challenging and computationally expensive.

7.1.3 Radiance Equation

Now we consider our general problem: surfaces and participating media together. Under the common simplifying assumptions of geometric optics, unpolarized light, time-invariance, and decoupled wavelengths, the essential equation of image synthesis that completely captures the distribution of light in a scene is the *time-invariant, gray radiance equation* [34]:

$$L(\mathbf{x}, \vec{\omega}) = \tau(\mathbf{x}, \mathbf{s}) \left[L_e(\mathbf{s}, \vec{\omega}) + \int_{\Theta} f_S(\mathbf{s}) L(\mathbf{s}, \vec{\omega}) d\vec{\omega} \right] + \int \tau(\mathbf{x}, \mathbf{a}) \left[L_e(\mathbf{a}, \vec{\omega}) + \int_{\Theta} p(\mathbf{a}, \vec{\omega}', \vec{\omega}) L(\mathbf{a}, \vec{\omega}) d\vec{\omega} \right] d\alpha \quad (7.4)$$

where $\mathbf{a} = \mathbf{x} - \alpha\vec{\omega}$, \mathbf{s} is the nearest surface point, and $L(\mathbf{x}, \vec{\omega})$ gives the radiance at point \mathbf{x} coming from direction $\vec{\omega}$. This equation addresses both surfaces and participating media and summarizes our rendering problem. It essentially says that to find the radiance at \mathbf{x} coming from direction $\vec{\omega}$, we find the nearest surface point \mathbf{s} , compute its outgoing radiance into $\vec{\omega}$ (its incoming radiance comes from all directions) and accumulate the radiance due to volume emission and inscattering along the way from \mathbf{s} to \mathbf{x} .

7.2 Liquid Surface Model

We represent a liquid surface (more precisely, a liquid-gas interface) with the following implicit surface:

$$\mathcal{L}(x, y, z) = \tau_{\mathcal{L}} \quad (7.5)$$

where $\tau_{\mathcal{L}}$ is a constant and $\mathcal{L} > \tau_{\mathcal{L}}$ is inside the liquid. The surface normal $\mathbf{n}_{\mathcal{L}}$ is taken in the direction pointing out of the liquid:

$$\mathbf{n}_{\mathcal{L}}(x, y, z) = -\frac{\nabla \mathcal{L}(x, y, z)}{|\nabla \mathcal{L}(x, y, z)|} \quad (7.6)$$

The implicit function \mathcal{L} is represented discretely on a finer resolution subgrid of our main simulation grid with samples at grid nodes.

We generate \mathcal{L} of the liquid surface from the summation of spherical field primitives \mathcal{L}_i associated with each reactive elements, i.e., the standard “blobby” approach to rendering surfaces using particles. However, to smooth out the surface, we use an ad hoc relaxation procedure where we diffuse \mathcal{L} to reduce the curvature $\kappa_{\mathcal{L}}$,

$$\kappa_{\mathcal{L}} = \nabla \cdot (\nabla \mathcal{L}_0 / |\nabla \mathcal{L}_0|) \quad (7.7)$$

in regions where the liquid velocity is near zero and the liquid surface is assumed to be flat.

7.2.1 Shading Model

Image synthesis requires a shading model for our surfaces, i.e., some approximation of the bidirectional scattering distribution function (BSDF). A wide variety of shading models have been developed by researchers in computer graphics that trade-off accuracy, expressiveness, and speed; Glassner [34] gives an excellent review of these models. Instead of a fixed model, shading functions can be modelled with custom procedures called *shaders*, programs that compute how a surface reflects light. For example, Hanrahan and Lawson [36] described a shading language for programming shaders based on the *shade trees* of Cook [12] and the *pixel stream editor* of Perlin [59].

While any shading model can be used for our surfaces, shading reactive liquids presents two basic challenges. First, the shading can rapidly and dramatically vary through time as it can depend on temperature and chemical concentrations. Thus, we need to animate parameters of our shading model. Unfortunately, most shading models are quite difficult to animate. Modifying one shading parameter may require modifying all other parameters to ensure the model is still “physically plausible”. Modifying a given parameter by the same amount may have very different effects on the shading depending on the value of the given parameter as well as the values of all other parameters. Ideally, we want a set of independent, normalized shading parameters that define a parameter space where the distance between points in this space correspond to the magnitude of change in the overall shading. Strauss [74] presented a shading model with such independent and normalized parameters, and we use this model.

The second challenge of shading reactive liquid surfaces is relating the shading to material properties such as temperature and chemical concentrations. Naturally, animators should be allowed to arbitrarily map fluid properties to shading parameters, and thus we need to define some way to do this. A procedural method is the most flexible approach, but implementing such a method is beyond the scope of this dissertation. Instead we use hard-coded mappings of fluid properties to shading parameters.

7.3 Participating Media

Most gases are invisible, and what we perceive of gaseous flow is typically through the motion of participating media such as dust, smoke, flame, and clouds. Liquids can contain participating media as well. Thus, we model the volume density $\mathcal{M}(x, y, z, t)$

of participating media.

7.3.1 Volume Density Model

Reactive elements track participating media. The challenge here is to model the wispy appearance of smoke and other media from ellipsoidal primitives. Stam [70] used “warped blobs” to achieve this effect by back-tracing through the velocity field. This approach is computationally expensive, however, requiring integration back through time to the initial distribution of primitives.

We experimented with another approach where elements are treated as participating medium sources where the medium is convected and diffused and eventually eliminated:

$$\frac{\partial \mathcal{M}}{\partial t} + \mathbf{v} \cdot \nabla \mathcal{M} = \alpha_{\mathcal{M}} \sum_i \mathcal{M}_i + d_{\mathcal{M}} \nabla^2 \mathcal{M} - \beta_{\mathcal{M}} \mathcal{M}$$

where $\alpha_{\mathcal{M}}$ is production rate, \mathcal{M}_i is a field function of the i th element, $\beta_{\mathcal{M}}$ is the loss rate, and $d_{\mathcal{M}}$ is the diffusion coefficient. While this approach produced wispy streams, we found it diffused the medium too much, smearing visual detail. Instead, we simply used the summation of stochastic field functions associated with each element. This approach makes it easy to add more visual detail where desired: simply use more elements.

7.3.2 Phase Functions

Shading participating media effectively amounts to choosing an appropriate phase function. Thus, we review various phase functions and argue for the one we have selected. We want a physically plausible phase function that is as general as possible without being too computationally expensive to compute.

As with any scattering function, a physically plausible phase function must satisfy two important properties. First, it must satisfy the symmetry relation (Helmholtz reciprocity):

$$p(\omega, \omega') = p(\omega', \omega)$$

Second, it must conserve energy. Thus, the following normalization condition must hold:

$$\frac{1}{4\pi} \int_S p(\omega, \omega') d\omega' = 1 \quad (7.8)$$

Assuming the phase function is isotropic, i.e., it depends only on angle $\alpha \in [0, \pi]$ between incident and outgoing directions (where $\alpha = 0$ means no change in direction and $\alpha = \pi$ means a total reversal of direction), equation (7.8) can be written as:

$$\frac{1}{2} \int_0^\pi p(\alpha) \sin \alpha d\alpha = 1 \quad (7.9)$$

A common convention when specifying phase functions that we adopt here is to use $a = \cos(\alpha)$ rather than α . Thus, equation (7.9) becomes:

$$\int_{-1}^1 p(a) da = 2$$

The simplest phase function satisfying reciprocity and normalization conditions is:

$$p_C(a) = 1$$

Blinn [5] presented some other simple phase functions, including an anisotropic function:

$$p_A(w, a) = 1 + wa$$

and a Lambert function:

$$p_L(a) = \frac{8}{3\pi} [\sin \alpha + (\pi - \alpha)a]$$

Most phase functions assume the participating medium consists of a homogeneous distribution of particles where the index of refraction is dominated by size of the particles. The scattering that occurs in such a medium is characterized by the ratio of particle size r to the wavelength λ of light involved (using 555nm as a characteristic wavelength). When particles are much smaller than the wavelength of light, $r \ll \lambda$, there is no appreciable scattering and the light is absorbed. When particles are much bigger, $r \gg \lambda$, then geometrical optics becomes a factor. Rayleigh scattering occurs when $r/\lambda < 0.05$ and is characteristic of cigarette smoke and dust. A phase function for Rayleigh scattering is [34]:

$$p_R(a) = \frac{3}{4} (1 + a^2)$$

Mie scattering is characteristic when particles are comparable to the wavelength of light such as for water droplets or fog. Nishita et al. [55] presented phase functions for Mie scattering with both sparse (hazy) and dense (murky) particle densities, i.e., hazy:

$$p_{M_h}(a) = 1 + 9 \left(\frac{1+a}{2} \right)^8$$

and murky:

$$p_{M_m}(a) = 1 + 50 \left(\frac{1+a}{2} \right)^{32}$$

Another approximation to the Mie functions is given by the Henyey-Greenstein phase function:

$$p_{HG}(g, a) = \frac{1 - g^2}{(1 - 2ga + g^2)^{3/2}} \text{ where } g \in [-1, 1]$$

where $g > 0$ is mostly backscattering, $g = 0$ is uniform scattering, and $g < 0$ is mostly forward scattering. This phase function was used by Fedkiw, Stam, and Jensen [19] in their recent work on animating smoke.

We use the two-term Henvey-Greenstein (TTHG) model which can approximate both Rayleigh and Mie scattering functions by linearly combining two ellipses of different magnitudes and eccentricities:

$$p_{TTHG}(r, g_1, g_2, a) = r \frac{1 - g_1^2}{(1 - 2g_1a + g_1^2)^{3/2}} + (1 - r) \frac{1 - g_2^2}{(1 - 2g_2a + g_2^2)^{3/2}}$$

This model is quite general, yet still reasonable to compute. (The fractional exponentiation can be avoided by using an approximation to the TTHG model [4].) In our implementation, the parameters r , g_1 , and g_2 are constants for each material type.

7.4 Rendering

We render our surface and volume density models using a bidirectional Monte Carlo ray-tracing algorithm based on a photon tracing method [41, 42, 43]. We describe our implementation of this method for completeness but do not make any significant modifications to it. This algorithm involves two passes. In the first pass, photons are traced from light sources to compute estimates of indirect illumination and caustics. In the second pass, rays are traced from the viewpoint into the scene, accounting for the effects of participating media and utilizing the information computed in the first pass.

7.4.1 Photon Tracing Pass

In the first pass of the rendering algorithm, we simulate the emission of photons from light sources and their propagation through the scene as they interact with participating media and surfaces. A photon is represented as a particle with position, (incident) direction, and power.

Photon Emission

In our implementation, we use a set of directional (infinite) light sources. Surfaces and participating media could act as light sources as well, but we have not implemented this. Photons are shot from the light sources with power:

$$P_{\text{photon}} = P_{\text{light}}/N_e$$

where N_e is the number of photons emitted. Note that more photons should be emitted from brighter lights than dim lights to ensure the power of all emitted photons is approximately even.

Photon Maps

Photon interactions with surfaces and volumes are stored in photon maps. A photon map records information about photon position, incoming power, and incident direction. We use three photon maps:

- the *caustic photon map* records photons having at least one specular reflection before hitting a diffuse surface,
- the *global photon map* records photons having at least one interaction before hitting a diffuse surface and is an approximate representation of the global illumination solution for all diffuse surfaces, and
- the *volume photon map* records photons interacting with participating media.

The notation of Heckbert [38] facilitates the expression of what interactions get stored. In this notation, L means emission from a light source, S is specular reflection or transmission, D is diffuse reflection or transmission, and V is volume scattering. Regular expressions are then built using these symbols, e.g., $()^*$ means

| Photon Map | Photon Path |
|------------|---------------|
| caustic | LS^+D |
| global | $L(S D V)^*D$ |
| volume | $L(S D V)^+V$ |

Table 7.1: Photon maps.

zero or more, $()^+$ means one or more, and $A|B$ means either A or B . Table 7.1 shows the regular expressions corresponding to the various photon maps.

Photon Interaction with Surfaces

When a photon hits a surface, a probabilistic method is used to determine whether it undergoes specular reflection, diffuse reflection, or is absorbed. The probability P_r of reflection is determined from the reflection coefficients in our shading models. The probability of absorption is thus $1 - P_r$. The power of a photon is appropriately diminished after reflecting off of a surface.

Photon Interaction with Participating Media

A photon can pass through a participating medium unaffected or it can interact with it (be scattered or absorbed). If a photon interacts with the medium and does not come directly from a light source, it gets recorded in the volume photon map. A cumulative probability density function, $F(\mathbf{x})$, is used to express the probability of a photon interacting with the medium:

$$F(\mathbf{x}) = 1 - \tau(\mathbf{x}_m, \mathbf{x})$$

where \mathbf{x}_m is the point where the photon entered the medium. If a photon interacts with the medium, the probability of it being scattered is given by the scattering

albedo. The new direction of the photon is chosen based on the phase function of the medium at \mathbf{x} .

7.4.2 Ray Tracing Pass

In the second pass of our rendering algorithm, we use a recursive ray-marching procedure. Rather than the original backward marching approach of Jensen and Christensen [43], we adopt the forward marching modification suggested by Fedkiw, Stam, and Jensen [19], i.e., we start at the viewpoint and march through any participating media until we hit a surface. We use an efficient voxel traversal algorithm to accelerate the ray marching [11].

Ray Marching Through Participating Media

Starting at \mathbf{x}_0 , the first point in Ω hit by a ray from the viewpoint through the image plane, we march along this ray in variable increments of length Δx_n for the n th step. Sampling at regularly spaced intervals here can result in strong aliasing artifacts, so it is essential to randomly perturb (jitter) this step size. Also, the step size can be adaptive, increasing the sampling frequency where variance is higher. The radiance at the n th step is given by:

$$L_n(\mathbf{x}_n, \vec{\omega}) = L_{n-1}(\mathbf{x}_{n-1}, \vec{\omega}) + \tau(\mathbf{x}_0, \mathbf{x}_n) \Delta x_n L_s(\mathbf{x}'_n, \vec{\omega})$$

where \mathbf{x}'_n is a random position in the n th ray segment and L_s is the fraction of the in-scattered radiance that is scattered in the direction $\vec{\omega}$:

$$L_s = \sigma_a(\mathbf{x}) L_e(\mathbf{x}, \omega) + \sigma_s(\mathbf{x}) \int L_i(\mathbf{x}, \omega) p(\mathbf{x}, \vec{\omega}', \vec{\omega}) d\vec{\omega}'$$

The in-scattered radiance L_i can be split into a single scattering term L_d and a multiple scattering term L_m :

$$L_i = L_d + L_m$$

The direct term L_d is computed using standard ray tracing. The multiple scattering term is approximated using the volume photon map [43]:

$$L_m(\mathbf{x}, \vec{\omega}) = \frac{1}{\sigma_s(\mathbf{x})} \sum_{p=1}^n \frac{\Phi_p(\vec{\omega}') p(\mathbf{x}, \vec{\omega}', \vec{\omega})}{4/3\pi r^3}$$

where $\Phi_p(\vec{\omega}')$ is the power of the p th photon and r is the radius of the smallest sphere containing the n closest photons. On the order of $n = 100$ seems to work well.

Ray-Surface Intersection

Once we hit a surface, the reflected radiance L is computed from the incoming illumination L_i as:

$$L(\mathbf{x}, \vec{\omega}) = \int f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L_i(\mathbf{x}, \vec{\omega}') \cos \theta_i d\omega'_i$$

where f_r is the BRDF. Following Jensen [42], the reflected radiance is decomposed into four components:

$$\begin{aligned} L(\mathbf{x}, \vec{\omega}) &= \int f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L_l(\mathbf{x}, \vec{\omega}') \cos \theta_i d\omega'_i \\ &+ \int f_{r,s}(\mathbf{x}, \vec{\omega}', \vec{\omega}) (L_c(\mathbf{x}, \vec{\omega}') + L_d(\mathbf{x}, \vec{\omega}')) \cos \theta_i d\omega'_i \\ &+ \int f_{r,d}(\mathbf{x}, \vec{\omega}', \vec{\omega}) L_c(\mathbf{x}, \vec{\omega}') \cos \theta_i d\omega'_i \\ &+ \int f_{r,d}(\mathbf{x}, \vec{\omega}', \vec{\omega}) L_d(\mathbf{x}, \vec{\omega}') \cos \theta_i d\omega'_i \end{aligned}$$

where L_l is direct illumination from the light sources, L_c is the indirect illumination from light sources via specular reflection or transmission (computed using the caustics photon map), and L_d is indirect illumination from light sources which has been reflected diffusely at least once (computed using the global photon map).

7.5 Results

In this section we present sequences of frames rendered using the techniques discussed in this chapter for several different animation scenarios. Figures 7.1–7.2 show a rendered sequence corresponding to the preview sequence shown in Figures 5.7–5.8. In this sequence the liquid surface is not rendered, only the liquid interior which is modelled to have a uniform participating medium with a pure blue hue. Some rendering artifacts are apparent. Figure 7.3 shows a sequence of a viscous liquid body flowing from an initial state where a hypothetical container around the liquid instantly vanishes (similar to the breaking dam sequence discussed previously). The liquid surface in this sequence is opaque.

Figures 7.4–7.5 show a sequence involving gaseous flow. The gas entirely fills the simulation domain. A layer of a participating medium (water vapor) lies at the bottom of the simulation domain. A point heat source at the bottom center of this domain creates a convection current which causes a gaseous jet to shoot upwards. This jet reaches the top of the domain creating a ring. Free-slip boundary conditions are used. Figure 7.6 shows another sequence with a participating medium in a gas. At the left side of the grid just above the top of the layer of participating medium at time $t = 0$ there is a solid obstacle (a block of solid cells) that is not rendered but its effects on the flow are apparent as the participating medium flows around this obstacle. This flow is caused by a force created in the bottom left corner, shooting

the gas into the solid obstacle. The gas flows around this obstacle, creating vortices as made apparent by the flow of the participating medium in the gas.

Figure 7.7 shows a sequence where a liquid blob, suspended above a cloud layer at time $t = 0$, falls through the cloud layer, punching a hole in it and carrying gas with it. The final frame of this sequence at time $t = 1.33$ shows the liquid blob hitting the bottom of the simulation grid (not rendered). Due to our crude approximation of the conditions at the liquid-gas interface, the gas does not properly slide off of the liquid as the liquid passes through. We believe this can be remedied by ad hoc modifications to the interface conditions. Despite this artifact, this sequence illustrates the integrated simulation of high density liquid moving through a low (assumed to be vanishingly small) density gas.

All of the images presented here as well as additional images and animations can be downloaded from the following web site:

<http://www.cs.ubc.ca/~gates/phd.html>

The interested reader is encouraged to visit this site to view the original images and to play the animations.



$t = 0$



$t = 0.33$



$t = 0.67$



$t = 1.0$

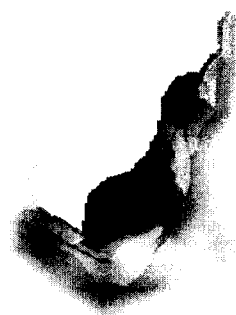


$t = 1.33$

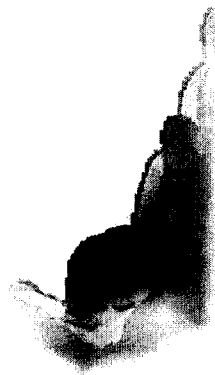


$t = 1.67$

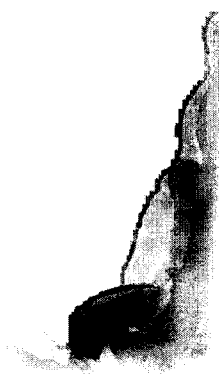
Figure 7.1: First set of frames from liquid drop sequence.



$t = 2.0$



$t = 2.33$

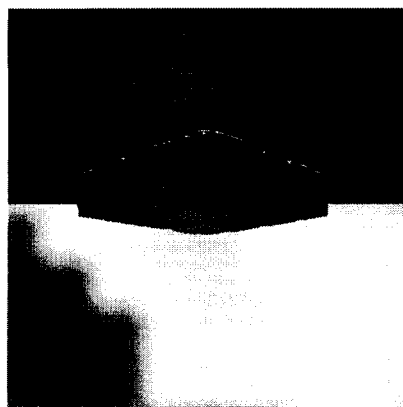


$t = 2.67$

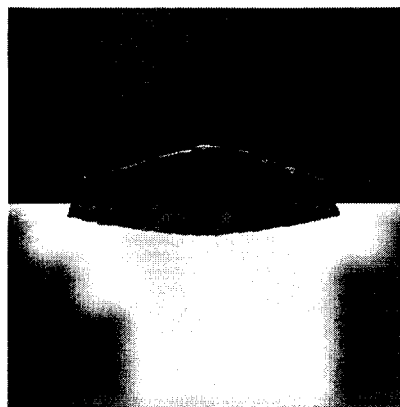


$t = 3.0$

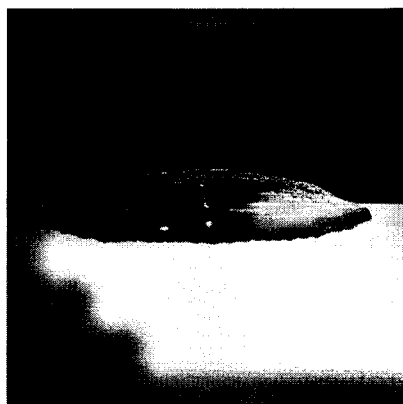
Figure 7.2: Second set of frames from liquid drop sequence.



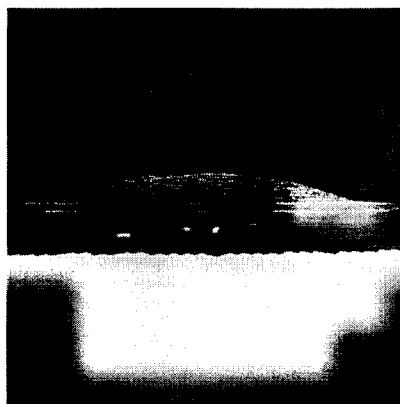
$t = 0$



$t = 0.33$



$t = 0.67$



$t = 1.0$

Figure 7.3: Frames from sequence.



$t = 0$



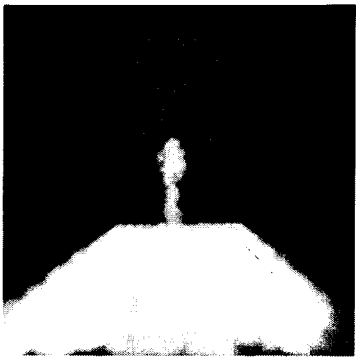
$t = 0.47$



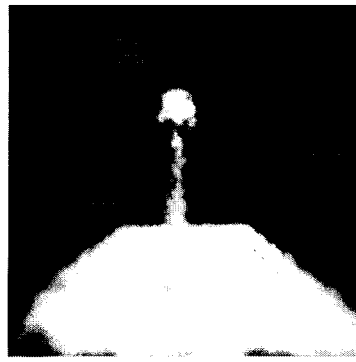
$t = 0.80$



$t = 1.3$

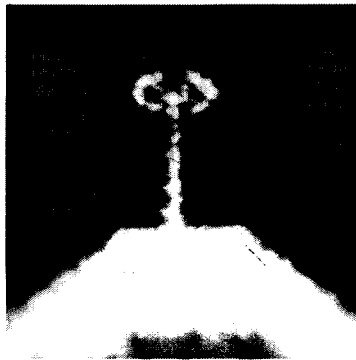


$t = 1.63$



$t = 2.3$

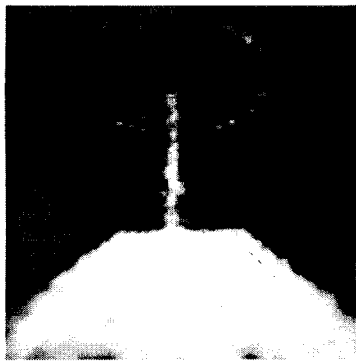
Figure 7.4: First set of frames from gas sequence.



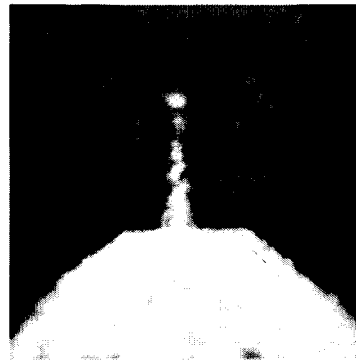
$t = 2.97$



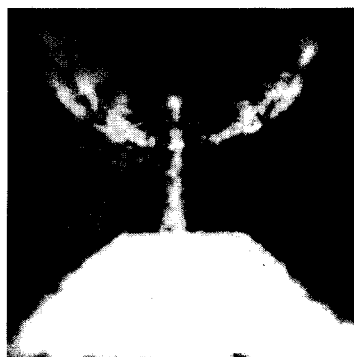
$t = 3.97$



$t = 4.97$

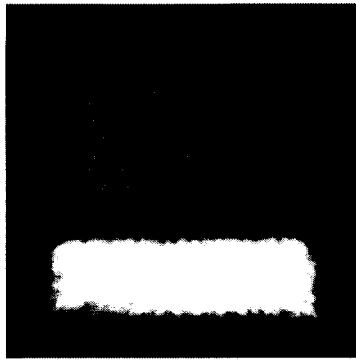


$t = 6.63$

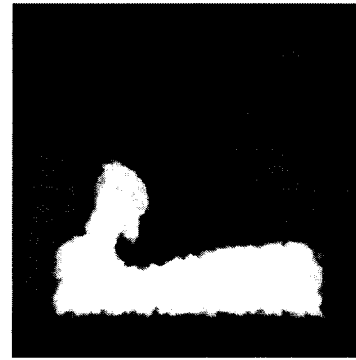


$t = 7.8$

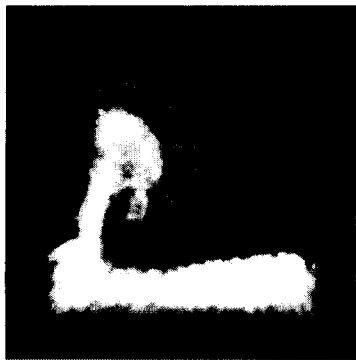
Figure 7.5: Second set of frames from gas sequence.



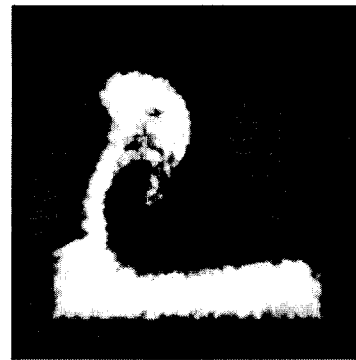
$t = 0$



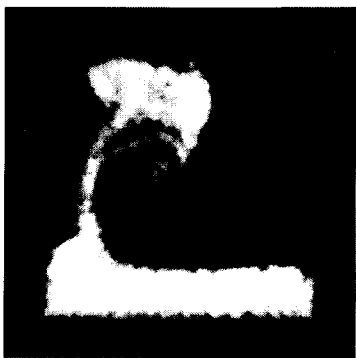
$t = 3.07$



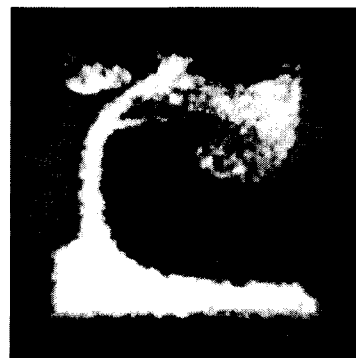
$t = 5.07$



$t = 7.07$

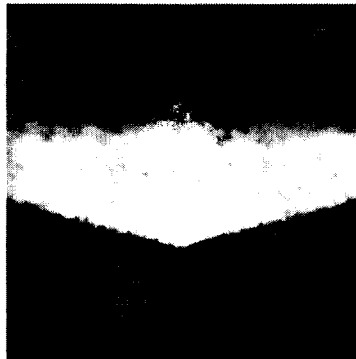


$t = 9.07$

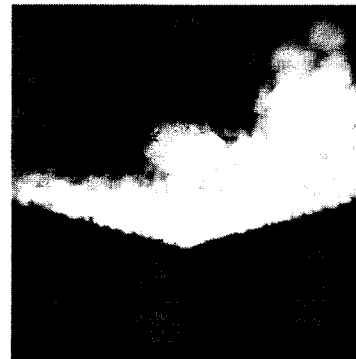


$t = 16.07$

Figure 7.6: Another gas sequence.



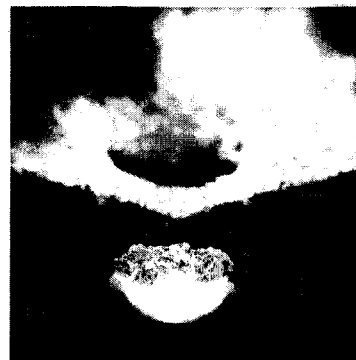
$t = 0$



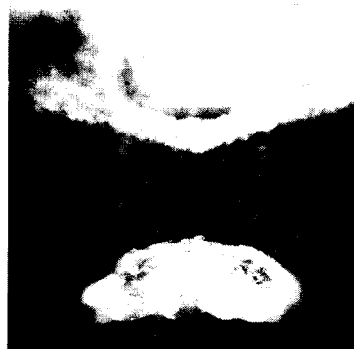
$t = 0.33$



$t = 0.67$



$t = 1.0$



$t = 1.33$

Figure 7.7: Liquid and gas simulation.

Chapter 8

Reactive Element Programs

In our methodology chemical reactions and other phenomena that affect the appearance and behaviour of fluid flow can be modelled using reactive element programs. These programs are procedures executed at every time step in the simulation to animate element properties. Since these properties are tightly coupled with our dynamic and shading models, simple programs can describe complex effects on the motion and appearance of fluid. In this chapter, we present a high level language for programming reactive elements, describe the implementation of a compiler for this language, and present a number of applications of reactive element programs.

8.1 Introduction

By default, reactive elements are passive and simply track the transport of material properties in the flow. However, as their name implies, reactive elements can play an active role in the simulation: each element can have an associated program that computes changes in its properties. A program can be associated with each material type. At each time step in the simulation, each element belonging to a “reactive”

material executes the program associated with that material.

A key aspect of this approach is that elements track dynamically simulated fluid properties, and thus changes in element properties feedback into the simulation. Naturally this approach requires a programming language for reactive elements. Ideally, this language should make it as easy as possible to program the desired effects. We present our language in Section 8.3, but first we discuss control surfaces and volumes.

8.2 Control Surfaces and Volumes

In our fluid model (c.f. Section 3.1) we defined a set of abstract *control surfaces* defined with the implicit functions \mathcal{S}_i and *control volumes* with density \mathcal{V}_i . The role of these control surfaces and volumes to provide additional “stimuli” for reactive elements, i.e., each element can “sense” the value of the implicit functions \mathcal{S}_i and volume densities \mathcal{V}_i . Thus, reactive element programs can compute responses to these surfaces and volumes, providing a general mechanism of influencing the procedural animation of element properties in a similar manner to our approach to influencing flow simulation with the control velocity field.

Control surfaces can be modelled using any of the myriad modelling methods for implicit surfaces. See the book by Bloomenthal [6] for a review of these methods. Likewise, control volumes can be modelled using any of the diverse methods for modelling density fields, e.g., interactive techniques [31, 82], procedural techniques [17, 27, 48, 57, 59, 60, 87], scan-converting geometric models onto a grid [45], and even using the output of computerized tomography (CT), magnetic resonance imaging (MRI), or positron emission tomography (PET) scans.

8.3 A Programming Language for Reactive Elements

We have developed a high level language specifically for programming reactive elements. This language is inspired by the success of the shading language described by Hanrahan and Lawson [36] for procedural models of shading functions based on the *shade trees* of Cook [12] and the *pixel stream editor* of Perlin [59]. As with their shading language, our language is based on a subset of the C programming language [46] and uses most of its syntax (i.e., it is one of the many “mini-C” languages). The distinguishing feature of our language is its set of predefined variables corresponding to element properties and parameters of our fluid model. Reactive element programs do not return a value, but rather cause side effects by modifying variables.

8.3.1 Data Types

To keep our programming language simple, we define just two data types:

- **float**: a typical floating point type, and
- **vector**: a 3-tuple of the **float** type.

The language does not support any fixed point type: the **float** type covers integers.

8.3.2 Predefined Variables

Table 8.1 lists all the predefined variables in our programming language for reactive elements. Note that some variables are “read-only”, i.e., their value can be accessed but not modified. All these variables are set with current values before the execution of a procedure.

| Variable | Type | Access | Description |
|----------|--------|--------|---|
| C_i | float | RW | concentration of chemical species i |
| D_i | float | RW | diffusion coefficient of chemical species i |
| H | float | RW | heat source rate |
| L_i | float | RW | loss rate of chemical species i |
| Q_i | float | RW | production rate of chemical species i |
| CS_i | float | R | implicit function of control surface i |
| CV_i | float | R | i density of control volume i |
| P | vector | R | position |
| T | float | R | temperature |
| V | vector | R | velocity |
| dt | float | R | current simulation time step |
| gravity | vector | R | gravity (uniform in arbitrary direction) |
| time | float | R | current time of simulation |

Table 8.1: Predefined variables (R = read only, RW = read and write).

| Operator | Description |
|-------------------|--------------------------|
| +, - | addition, subtraction |
| *, / | multiplication, division |
| ^ | exponentiation |
| &&, | logical and, logical or |
| ! | logical not |
| <, <=, >, >= | relational operators |
| == | equals to |
| =, +=, -=, *=, /= | assignment |

Table 8.2: Operators in order of increasing precedence.

| Function | Description |
|---------------------------------------|-------------------------------|
| <code>float abs(float)</code> | absolute value |
| <code>float acos(float)</code> | arc cosine |
| <code>float asin(float)</code> | arc sine |
| <code>float atan(float)</code> | arc tangent |
| <code>float cos(float)</code> | cosine |
| <code>float length(vector)</code> | magnitude of vector |
| <code>float log(float)</code> | logarithm |
| <code>float max(float,float)</code> | maximum of arguments |
| <code>float min(float,float)</code> | minimum of arguments |
| <code>vector normalize(vector)</code> | normalizes vector |
| <code>float pow(float,float)</code> | power |
| <code>float random()</code> | random number in range [0, 1] |
| <code>float sin(float)</code> | sine |
| <code>float sqrt(float)</code> | square root |
| <code>float tan(float)</code> | tangent |

Table 8.3: Built-in functions.

8.3.3 Operators

As shown in Table 8.2, the language supports most of the standard operators on floats in the C programming language (with the same precedence and associativity rules). Arithmetic operators (addition, subtraction, multiplication, and division) are also defined for vectors.

8.3.4 Built-in Functions

Our programming language also includes a number of built-in functions which are listed in Table 8.3.4. Most of these functions are standard routines in the C math library.

8.4 Compilation

Execution of reactive element programs can easily be the limiting step in the entire simulation algorithm: every element can execute a procedure of arbitrary computational complexity at every time step. Thus, rapid execution of procedures is critical. Interpretation would be extremely inefficient as every execution of a procedure would require lexical scanning, parsing, and type checking. Instead, we first compile procedures to a lower level intermediate form before executing them. This intermediate form consists of an array of “byte codes”, i.e., simple instructions that are executed on a stack-based “virtual machine” which processes byte codes by pushing and popping their operands and results on and off a stack. Figure 8.1 shows the source code for a procedure, and Figure 8.2 shows the corresponding intermediate code. In their compiled form, procedures can be executed very efficiently: all syntax and type checking has been performed, all symbols have been resolved (no symbol lookup is necessary), the overhead of function calls is essentially eliminated, and the array facilitates caching by the CPU.

Our compiler implementation uses standard techniques [1] and tools (*flex* and *bison*) [47]. A lexical scanner generated using *flex* converts procedure source code to tokens. A parser generated with *bison* generates a parse tree from these tokens. After type checking and symbol resolution, the parse tree is walked to generate the byte codes. The implementation of the “virtual machine” is straightforward, essentially involving a single “switch” statement with “case” statements for each byte code.

```
reaction A () {  
  if (T > 1)  
  {  
    Q1 = 100;  
  }  
  else  
  {  
    Q1 = 0;  
  }  
}
```

Figure 8.1: Reaction A source.

```
reaction A  
  floats:  
    1  1.000000  
    2  100.000000  
    3  0.000000  
  code:  
    0  push_prop temp  
    2  push_f 1  
    4  gt  
    5  ifz 13  
    7  push_f 2  
    9  pop_prop prod1  
   11  jump 17  
   13  push_f 3  
   15  pop_prop prod1  
   17  return
```

Figure 8.2: Reaction A compiled.

```

reaction B () {
    float k = 1;

    if ((C1 > 0) && (C2 > 0))
    {
        L1 = k;
        L2 = k;
        Q0 = k;
        H = 10;
    }
    else
    {
        Q0 = 0;
    }
}

```

Figure 8.3: Reaction B source.

```

reaction B
floats:
    1  0.000000
    2  1.000000
    3  0.000000
    4  0.000000
    5  10.000000
    6  0.000000
code:
    0  push_f 2
    2  pop_f 1
    4  push_prop chem1
    6  push_f 3
    8  gt
    9  push_prop chem2
   11  push_f 4
   13  gt
   14  and
   15  ifz 35
   17  push_f 1
   19  pop_prop loss1
   21  push_f 1
   23  pop_prop loss2
   25  push_f 1
   27  pop_prop prod0
   29  push_f 5
   31  pop_prop heat
   33  jump 39
   35  push_f 6
   37  pop_prop prod0
   39  return

```

Figure 8.4: Reaction B compiled.

8.5 Results

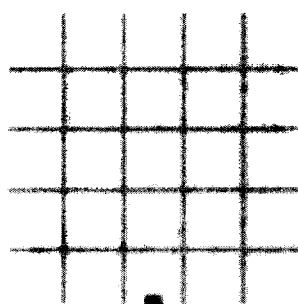
Simple reactive element programs can produce interesting results. In the following examples, a $50 \times 1 \times 50$ grid (i.e., a “vertical slice”) is used. The entire domain is filled with liquid, and free-slip boundary conditions are used. Chemical concentrations are mapped to colour channels where the red channel $r = 1 - C_0$, the green channel $g = 1 - C_1$, and the blue channel $b = 1 - C_2$ where the colour channels are normalized on the range $[0, 1]$. Recall that the chemical concentrations are also normalized on the range $[0, 1]$. The simulation clamps the concentrations at minimum and maximum values, so they are guaranteed to stay in this range.

Figure 8.1 shows the source code listing for Reaction A, and Figure 8.2 shows the corresponding code after compilation. The compiled code is executed at every time step in the simulation. This reaction program simply generates Chemical 1 if the temperature exceeds a threshold value. Figures 8.5–8.6 show the frames from a sequence using Reaction A. At time $t = 0$ in this sequence, Chemical 0 is distributed in a grid pattern at maximum concentration $C_0 = 1$ to help visualize the flow (it is inert), and a point heat source is instantaneously generated at the bottom center of the grid triggering the production of Chemical 1. This heat source creates a convection current which rises upwards, triggering the production of Chemical 1 as the temperature is raised above the threshold temperature defined in Reaction A. Note that the inert Chemical 0 diffuses over time despite its diffusion coefficient being zero (the default value). Thus, this diffusion is entirely numerical and is a byproduct of using the semi-Lagrangian integration scheme for convection (as well as using fairly coarse grids).

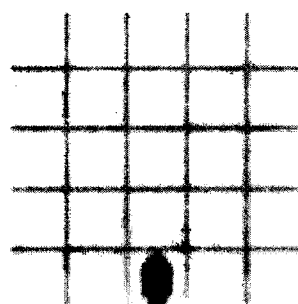
Figure 8.3 shows the source code listing for a slightly more complicated program Reaction B. Figure 8.4 shows the corresponding code after compilation. In

this program, a reaction is defined when Chemical 1 and Chemical 2 come into contact. When they do, they are converted via an implied combination reaction to Chemical 0 by triggering the loss of Chemical 1 and Chemical 2 and the production of Chemical 0. This program defines the reaction to generate heat. Figures 8.7–8.8 show frames from an animation sequence using Reaction B. At time $t = 0$, a layer of Chemical 1 sits just above a layer of Chemical 2. A nonzero velocity value is set at a grid point in the center of the region between the two layers, i.e., the liquid is slightly stirred. This stirring mixes the two layers which start to react. The reaction generates heat, creating a convection current which further mixes the chemicals and propagates the reaction.

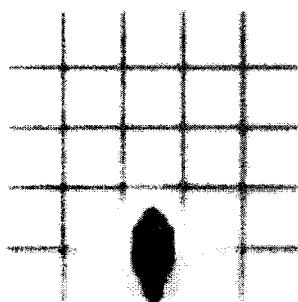
These examples illustrate effects that would be difficult to achieve using existing animation techniques. Our approach generalizes reaction-diffusion techniques used in computer graphics [79, 85] as our system simulates advection-reaction-diffusion. More fundamentally, reactive element programs couple the simulation of chemical and thermal reactions with our fluid flow simulation, allowing the modelling of complex visual effects with simple programs.



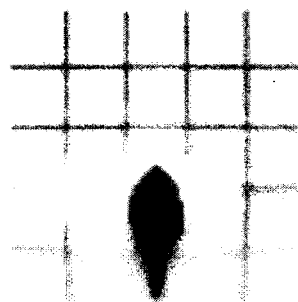
$t = 0$



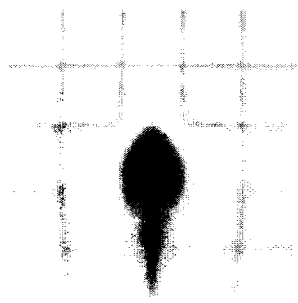
$t = 0.43$



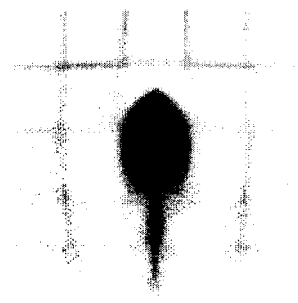
$t = 0.86$



$t = 1.30$

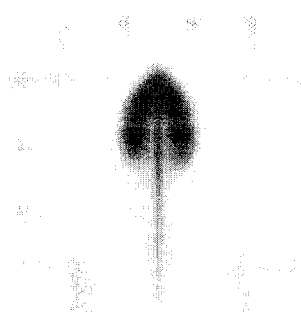


$t = 1.73$

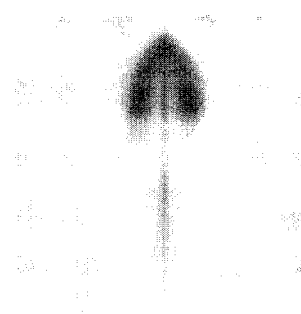


$t = 2.166$

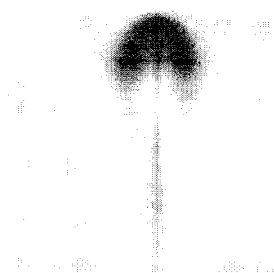
Figure 8.5: First set of frames from Reaction A sequence.



$t = 2.59$



$t = 3.03$



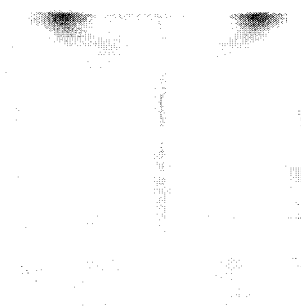
$t = 3.46$



$t = 3.90$



$t = 4.33$



$t = 4.76$

Figure 8.6: Second set of frames from Reaction A sequence.



$t = 0$



$t = 0.5$



$t = 0.63$



$t = 0.73$



$t = 0.80$



$t = 0.87$

Figure 8.7: First set of frames from Reaction B sequence.



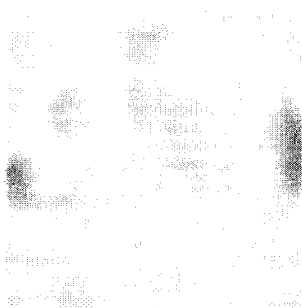
$t = 1.0$



$t = 1.13$



$t = 1.20$



$t = 1.27$

Figure 8.8: Second set of frames from Reaction B sequence.

Chapter 9

Conclusions

9.1 Summary

We have presented a general methodology for the computer animation of fluids based on the numerical simulation of the Navier-Stokes equations in three dimensions. This methodology addresses three basic challenges of animating fluids:

1. Integrated Animation of Fluids

We have presented a coherent framework for the integrated animation of fluids that couples a numerical fluid flow simulation method with both a novel procedural animation method for chemical and thermal reactions and a novel interactive technique for controlling the simulated flow. The key to this framework is the use of a set of scalar and vector fields as a unified representation of physical properties, shape models (implicit surfaces for liquids and volume density functions for participating media), shading attributes, and control parameters. Using this framework as a conceptual basis, our methodology allows the integrated animation of both the motion and appearance of both liquids and gases containing participating media and reactive chemical species.

2. Animation of Reactive Flow

Our methodology incorporates a procedural animation method for modelling chemical and thermal reactions that affect the appearance and behaviour of fluids. This method is based on a language specifically designed for programming these reactive effects—an open-ended approach that allows the modelling of arbitrary reactions. We have shown that simple programs can produce interesting results.

3. Control of Flow Simulation

Our methodology incorporates a novel method of controlling incompressible flow simulation where the animator can specify the desired flow using streamtube primitives. This method allows the modelling of a control velocity at every point in the flow as well as the degree to which the simulated flow should be constrained to match it. As the flow associated with the primitives is incompressible, our simulation method can compute incompressible flow continuously throughout the domain where the fluid velocity within the streamtube primitives corresponds to what the animator specified. This approach gives a much greater degree of control than explicitly modelling body forces.

9.2 Analysis

Our methodology has a number of strengths and weaknesses. It advances the state of the art in the computer animation of fluid flow by providing a coherent set of techniques for the integrated animation of a significantly larger class of fluid phenomena than previously addressed. It includes novel techniques for controlling fluid flow

simulation and for animating chemical and thermal reactions in fluids. It facilitates the animation of scenarios involving goal-directed fluid motion and reactive flow—scenarios that would be painstaking to animate using existing techniques. Perhaps the most significant contribution of this dissertation is our method of controlling flow simulation: we believe this method provides an intuitive and powerful approach to “directing” simulated flow to “perform” as desired.

Our methodology has several limitations which should be noted. It does not address turbulent flow or surface tension. As discussed in Section 5.13, our simulation algorithm is stable subject to a CFL condition which limits the maximum time step that can be taken. The use of semi-Lagrangian integration results in significant numerical diffusion: a trade-off we accept in exchange for stability. We have oversimplified the liquid-gas interface conditions which can result in visible artifacts. Our liquid surface representation is not sufficiently smooth unless a large number of reactive elements is used, and an adequate number is difficult to determine without experimentation. The use of a fixed, uniform grid limits the range of scales that can be modelled effectively at a given time.

We also note that a potential drawback of our methodology is the complexity of its implementation which involves a three-dimensional fluid flow solver that can handle free surfaces, a two-pass volume raytracer, and a compiler (to byte codes) for a C-like programming language. Such an implementation involves many subtleties; we hope we have provided sufficient detail for those interested in the implementation issues.

9.3 Future Work

The limitations of our methodology discussed in the previous section can be addressed by future work. A more accurate model of the liquid-gas interface conditions could be used but might require a different and possibly more computationally expensive flow simulation algorithm. A multi-resolution fluid representation and flow solver would overcome the limitations of using a regular grid. Recent work by Foster and Fedkiw [22] has shown that the use of level sets in conjunction with particles can be an effective means of tracking a free surface that is very smooth. This approach appears quite promising and could be adapted to work with our methodology.

Future work can also extend our methodology in a number of interesting directions. Other types of flow primitives besides streamtubes could be developed. Animated flow primitives may have useful applications. Complex interactions between fluids and solids such as those involved in erosion and weathering could be modelled in terms of mechanical, chemical, and thermal reactions. While specific weather and erosion phenomena have been effectively addressed using ad hoc techniques (as discussed in Chapter 2), a general approach has not yet been explored.

The computer animation of fluids remains one of the most difficult tasks in computer graphics. Many challenges must be tackled to alleviate this difficulty. The alluring beauty of flowing fluid ensures they will be.

Notation

All vectors and vector-valued functions are in boldface. Greek symbols are listed first, followed by Roman symbols.

| | |
|------------------------|---|
| α_c | degree of velocity control |
| β | thermal buoyancy scaling factor |
| Γ | boundary of domain Ω |
| $\delta\tau$ | cell dimension (grid spacing) |
| δt | time step |
| ϵ | error tolerance |
| $\kappa_{\mathcal{L}}$ | curvature of liquid surface |
| λ_i | diffusion coefficient of chemical species i |
| λ_T | thermal diffusivity |
| μ | dynamic viscosity |
| ν | kinematic viscosity |
| ρ | density |
| σ_a | absorption coefficient |
| σ_s | scattering coefficient |
| σ_t | extinction coefficient |
| τ | transmittance |

| | |
|----------------------------|---|
| ϕ | phase (gas, liquid, or solid) |
| Φ | material type |
| ω | vorticity |
| Ω | spatial domain |
| \mathbf{b} | body forces |
| C_i | concentration of chemical species i |
| \mathbf{f} | body forces other than gravity |
| \mathbf{g} | gravity |
| \mathcal{I} | influence field |
| L | radiance |
| \mathcal{L} | implicit function for free liquid surface |
| L_i | loss rate of chemical species i |
| \mathcal{M} | density of participating media |
| $\mathbf{n}_{\mathcal{L}}$ | normal to liquid surface |
| N_x | number of cells in x -direction |
| N_y | number of cells in y -direction |
| N_z | number of cells in z -direction |
| p | pressure |
| q | heat source |
| Q_i | production rate of chemical species i |
| \mathcal{S}_i | implicit function for control surface i |
| t | time |
| \mathbf{T} | stress tensor |
| T | temperature |
| T_∞ | ambient temperature |

u velocity component in x-direction
 v velocity component in y-direction
 \mathbf{v} velocity
 \mathbf{v}^* intermediate velocity
 \mathbf{v}_c control velocity
 \mathcal{V}_i density field of control volume i
 w velocity component in z-direction

Glossary

advection transport by flow

BRDF bidirectional reflectance distribution function

BSDF bidirectional surface scattering distribution function

BTDF bidirectional transmittance distribution function

cell an axis-aligned volume element of the simulation grid

combustion rapid oxidation reaction

compressible flow flow where fluid density changes in response to applied pressure

dynamic viscosity a measure of the resistance of a fluid to flow

Euler equations equations for inviscid, incompressible flow

Euler's method an explicit numerical integration scheme

Eulerian fluid flow is described with respect to a fixed frame of reference (cf. Lagrangian)

fluid substance that cannot resist a shear force at rest; gas or liquid

free surface liquid-gas interface

incompressible flow idealized flow where changes in applied pressure do not affect fluid density

inviscid flow idealized flow with zero viscosity

irrotational flow idealized flow with zero vorticity

kinematic viscosity ratio of dynamic viscosity to density

laminar flow relatively low speed flow where fluid particles follow smooth trajectories (cf. *turbulent flow*)

Lagrangian description of fluid motion where individual fluid particles are followed (cf. *Eulerian*)

Navier-Stokes equations the basic equations of viscous fluid flow first derived by Claude Navier in 1822 and later rederived on a more sound theoretical basis by George Stokes in 1845

pathline curve traced by a particular fluid particle during a specified time interval, e.g., the curve resulting from a long exposure photograph of a luminescent particle (cf. *streakline* and *streamline*)

reactive element a volume primitive defined in this dissertation for modelling chemical reactions in fluid flow and visualizing unsteady flow fields

solenoidal vector field a vector field with zero divergence

streakline curve consisting of all fluid particles that have passed through a given point during a specific interval of time, e.g., the curve traced by the continuous injection of dye at a given point (cf. *pathline* and *streamline*)

streamline a curve everywhere tangent to the instantaneous velocity vector; for

steady flow, a streamline is the same as a *pathline* and *streakline*

thermal buoyancy the tendency for warmer fluid to rise and colder fluid to sink

transmittance in radiation transfer, the fraction of incoming radiation that is

transmitted into or through a medium

turbulent flow relatively high speed flow characterized by a high degree of irreg-

ularity and vorticity (cf. *laminar flow*)

unsteady flow time-varying flow

viscosity see *dynamic viscosity*

vorticity the curl of a velocity field; a measure of local rotation in a fluid flow

Bibliography

- [1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Mass., 1986.
- [2] Rutherford Aris. *Vectors, Tensors, and the Basic Equations of Fluid Mechanics*. Dover, New York, 1989.
- [3] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, Cambridge, 1967.
- [4] Philippe Blasi, Bertrand Le Saëc, and Christophe Schlick. A rendering algorithm for discrete volume density objects. *Computer Graphics Forum (Eurographics '93)*, 12(3):201–210, 1993.
- [5] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.
- [6] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan-Kaufmann, San Francisco, CA, 1997.
- [7] Jim X. Chen. *Physically-Based Modeling and Real-Time Simulation of Fluids*. Ph.D. thesis, University of Central Florida, 1995.
- [8] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, and J. Michael Moshell. Real-time fluid simulation in a dynamic virtual environment. *IEEE Computer Graphics & Applications*, 17(3):52–61, May–June 1997. ISSN 0272-1716.
- [9] N. Chiba, S. Ohkawa, K. Muraoka, and M. Miura. Two-dimensional visual simulation of flames, smoke and the spread of fire. *The Journal of Visualization and Computer Animation*, 5(1):37–54, January–March 1994.
- [10] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2:12–26, 1967.
- [11] John G. Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

- [12] Robert L. Cook. Shade trees. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.
- [13] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 287–290. ACM SIGGRAPH, Addison Wesley, August 1995.
- [14] Julie Dorsey, Alan Edelman, Justin Legakis, Henrik Wann Jensen, and Hans K hling Pedersen. Modeling and rendering of weathered stone. *Proceedings of SIGGRAPH 99*, pages 225–234, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.
- [15] Julie Dorsey, Hans K hling Pedersen, and Pat Hanrahan. Flow and changes in appearance. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 411–420. Addison Wesley, August 1996.
- [16] David S. Ebert. *Solid Spaces: A unified approach to describing object attributes*. Ph.D. thesis, Ohio State University, 1991.
- [17] David S. Ebert. Design and animation of volume density functions. *The Journal of Visualization and Computer Animation*, 4(4):213–232, October–December 1993.
- [18] David S. Ebert, Wayne E. Carlson, and Richard Parent. Solid spaces and inverse particle systems for controlling the animation of gases and fluids. *The Visual Computer*, 10(4):179–190, 1994.
- [19] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22. ACM Press / ACM SIGGRAPH, August 2001.
- [20] Kurt W. Fleischer. *A Multiple-Mechanism Development Model for Defining Self-Organizing Geometric Structures*. Ph.D. thesis, California Institute of Technology, Pasadena, California, 1995.
- [21] Kurt W. Fleischer, David Laidlaw, Bena Currin, and Alan H. Barr. Cellular texture generation. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 239–248. Addison Wesley, August 1995.
- [22] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 23–30. ACM Press / ACM SIGGRAPH, August 2001.

- [23] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.
- [24] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. In *Graphics Interface '96*, pages 204–212, May 1996. ISBN 0-9695338-5-3.
- [25] Nick Foster and Dimitri Metaxas. Controlling fluid animation. In *Proceedings of Computer Graphics International '97*, pages 178–188, June 1997.
- [26] Nick Foster and Dimitri Metaxas. Modeling the motion of hot, turbulent gas. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 181–188. Addison Wesley, August 1997.
- [27] Alain Fournier and John Amanatides. 3-D texture mapping. In *Intl. Conf. on Engineering and Computer Graphics*. Beijing, China, August 1984.
- [28] Alain Fournier, Donald Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982.
- [29] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 75–84, August 1986.
- [30] Deborah R. Fowler, H. Meinhardt, and Przemyslaw Prusinkiewicz. Modeling seashells. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 379–387, July 1992.
- [31] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 267–274, July 1991.
- [32] Manuel Noronha Gamito, Pedro Faria Lopes, and Mário Rui Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. *Computer Animation and Simulation '95*, pages 2–15, September 1995.
- [33] William F. Gates. Interactive flow field modeling for the design and control of fluid motion in computer animation. Masters thesis, University of British Columbia, 1994.
- [34] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan-Kaufmann, San Francisco, CA, 1995.
- [35] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. SIAM, 1998.

- [36] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 289–298, August 1990.
- [37] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids*, (8):2182–2189, 1965.
- [38] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):145–154, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.
- [39] T. L. Hilton and P. K. Egbert. Vector fields: an interactive tool for animation, modeling and simulation with physically based 3D particle systems and soft objects. In *Computer Graphics Forum*, volume 13, pages 329–338. Eurographics, Basil Blackwell Ltd, 1994. Eurographics '94 Conference issue.
- [40] Masa Inakage. A simple model of flames. In *Proceedings of Computer Graphics International '89*, pages 71–81. Springer-Verlag, 1989.
- [41] Henrik W. Jensen and Niels J. Christensen. Efficiently rendering shadows using the photon map. *Compugraphics '95*, pages 285–291, December 1995. ISBN 972-8342-00-4.
- [42] Henrik Wann Jensen. Global illumination using photon maps. *Eurographics Rendering Workshop 1996*, pages 21–30, June 1996. ISBN 3-211-82883-4. Held in Porto, Portugal.
- [43] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Proceedings of SIGGRAPH 98*, pages 311–320, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.
- [44] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 49–57, August 1990.
- [45] Arie Kaufman, Daniel Cohen, and Roni Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, July 1993.
- [46] Brian Kernighan and Dennis Ritchie. *The C Programming Language, Second Edition*. Prentice Hall, Englewood Cliffs, NJ, 1988.

- [47] John Levine, Tony Mason, and Doug Brown. *lex & yacc*. O'Reilly and Associates, 1992.
- [48] John-Peter Lewis. Algorithms for solid noise synthesis. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 263–270, July 1989.
- [49] N. L. Max. Vectorized procedural models for natural terrain: Waves and islands in the sunset. volume 15, pages 317–324, August 1981.
- [50] Oleg Mazarak, Claude Martins, and John Amanatides. Animating exploding objects. *Graphics Interface '99*, pages 211–218, June 1999. ISBN 1-55860-632-7.
- [51] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.
- [52] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 41–50, July 1989.
- [53] Kenji Nagashima. Computer generation of eroded valley and mountain terrains. *The Visual Computer*, 13:456–464, 1997.
- [54] Michael Neff and Eugene L. Fiume. A visual model for blast waves and fracture. *Graphics Interface '99*, pages 193–202, June 1999. ISBN 1-55860-632-7.
- [55] Tomoyuki Nishita, Yasuhiro Miyawaki, and Eihachiro Nakamae. A shading model for atmospheric scattering considering luminous intensity distribution of light sources. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):303–310, July 1987. Held in Anaheim, California.
- [56] James F. O'Brien and Jessica K. Hodgins. Dynamic simulation of splashing fluids. In *Computer Animation '95*, pages 198–205, April 1995.
- [57] Darwyn R. Peachey. Solid texturing of complex surfaces. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 279–286, July 1985.
- [58] Darwyn R. Peachey. Modeling waves and surf. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 65–74, August 1986.
- [59] Ken Perlin. An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 287–296, July 1985.

- [60] Ken Perlin and Eric M. Hoffert. Hypertexture. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 253–262, July 1989.
- [61] Chris. H. Perry and Rosalind W. Picard. Synthesizing flames and their spreading. In *Proceedings of the Fifth Eurographics Workshop on Animation and Simulation*. Eurographics, September 1994.
- [62] William T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.
- [63] P. Roudier, B. Peroche, and M. Perrin. Landscapes synthesis achieved through erosion and deposition process simulation. In *Eurographics '93*, pages 375–383, Oxford, UK, 1993. Eurographics, Blackwell Publishers.
- [64] Georgios Sakas. Modeling and animating turbulent gaseous phenomena using spectral synthesis. *The Visual Computer*, 9(4):200–212, January 1993.
- [65] Robert G. Scharein and Kellogg S. Booth. Interactive knot theory with Knot-Plot. In *Multimedia Tools for Communicating Mathematics*, pages 277–290. Springer-Verlag, Berlin, Heidelberg, 2002.
- [66] Mikio Shinya and Alain Fournier. Stochastic motion: Motion under the influence of wind. In *Eurographics '92*, pages 119–128. Eurographics, 1992.
- [67] Karl Sims. Particle animation and rendering using data parallel computation. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413, August 1990.
- [68] Jos Stam. Stable fluids. *Proceedings of SIGGRAPH 99*, pages 121–128, August 1999.
- [69] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 369–376, August 1993.
- [70] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 129–136. ACM SIGGRAPH, Addison Wesley, August 1995.
- [71] A. Staniforth and J. Cote. Semi-lagrangian integration schemes for atmospheric models – a review. *Monthly Weather Review*, 119:2206–2223, 1992.

- [72] J. Steinhoff and D. Underhill. Modificaiton of the euler equations for “vorticity confinement”: Application to the computation of interacting vortex rings. *Physics of Fluids*, 6(8):2738–2744, 1994.
- [73] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. *Graphics Interface '99*, pages 203–210, June 1999. ISBN 1-55860-632-7.
- [74] Paul S. Strauss. A realistic lighting model for computer animators. *IEEE Computer Graphics & Applications*, 10(6):56–64, November 1990.
- [75] Demetri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models (from goop to glop). In *Proceedings of Graphics Interface '89*, pages 219–226, June 1989.
- [76] David Tonnesen. Modeling liquids and solids using thermal particles. In *Proceedings of Graphics Interface '91*, pages 255–262, June 1991.
- [77] Pauline Y. Ts'o and Brian A. Barsky. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics*, 6(3):191–214, 1987.
- [78] Alan Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, August 1952.
- [79] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.
- [80] Marcelo Walter, Alain Fournier, and Daniel Menevaux. Integrating shape and pattern in mammalian models. *Proceedings of SIGGRAPH 2001*, pages 317–326, August 2001. ISBN 1-58113-292-1.
- [81] Marcelo Walter, Alain Fournier, and Mark Reimers. Clonal mosaic model for the synthesis of mammalian coat patterns. *Graphics Interface '98*, pages 82–91, June 1998. ISBN 0-9695338-6-1.
- [82] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In *1995 Symposium on Interactive 3D Graphics*, pages 151–156. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.
- [83] Henrik Weimer and Joe Warren. Subdivision schemes for fluid flow. *Proceedings of SIGGRAPH 99*, pages 111–120, August 1999.

- [84] Jakub Wejchert and David Haumann. Animation aerodynamics. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 19–22, July 1991.
- [85] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.
- [86] Patrick Witting. Computational fluid dynamics in a traditional animation environment. *Proceedings of SIGGRAPH 99*, pages 129–136, August 1999.
- [87] Steven P. Worley. A cellular texturing basis function. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996.
- [88] Larry Yaeger, Craig Upson, and Robert Myers. Combining physical and visual simulation – creation of the planet jupiter for the film “2010”. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):85–93, August 1986.
- [89] Gary D. Yngve, James F. O'Brien, and Jessica K. Hodgins. Animating explosions. *Proceedings of SIGGRAPH 2000*, pages 29–36, July 2000. ISBN 1-58113-208-5.

Appendix A

Free Surface Conditions

When simulating liquids, appropriate velocity values on the faces between liquid and gas cells must be set so that the liquid cells neighboring gas cells have zero divergence. Except for the case of exactly one gas neighbor, there is no unique way to do this. Thus, reasonable assumptions about the flow in these cells must be made. Here we show how we set these velocity values for all 63 configurations of gas neighbors around a liquid cell. The notation used in the source code listing here is consistent with the notation used in the rest of this dissertation with one exception: we use H here for the grid spacing instead of $\delta\tau$. While the grid is uniform in each direction, the conditions are formulated here to facilitate the conversion to different grid spacings (δx , δy , δz) in each coordinate direction. The neighbor flag NF indicates which neighbors of the liquid cell (i, j, k) are gas cells. The neighbor flag uses a six digit binary number to represent the $-x$, $+x$, $-y$, $+y$, $-z$, and $+z$ gas neighbors respectively, e.g., 100000 indicates the liquid cell has exactly one neighbor in the $-x$ direction.

The conditions we use are as follows:

```

//
// one gas neighbor
//

case NF_100000:
    U(i-1,j,k) = U(i,j,k) + H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_010000:
    U(i,j,k) = U(i-1,j,k) - H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_001000:
    V(i,j-1,k) = V(i,j,k) + H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_000100:
    V(i,j,k) = V(i,j-1,k) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_000010:
    W(i,j,k-1) = W(i,j,k) + H*(
        (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_000001:
    W(i,j,k) = W(i,j,k-1) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
    break;

//
// two gas neighbors (opposite)
//

case NF_110000:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
        (V(i,j,k)-V(i,j-1,k))/H +
        (W(i,j,k)-W(i,j,k-1))/H;
    U(i-1,j,k) += 0.5*H*div;
    U(i,j,k) -= 0.5*H*div;

```

```

        break;
case NF_001100:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
          (V(i,j,k)-V(i,j-1,k))/H +
          (W(i,j,k)-W(i,j,k-1))/H;
    V(i,j-1,k) += 0.5*H*div;
    V(i,j,k) -= 0.5*H*div;
    break;
case NF_000011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
          (V(i,j,k)-V(i,j-1,k))/H +
          (W(i,j,k)-W(i,j,k-1))/H;
    W(i,j,k-1) += 0.5*H*div;
    W(i,j,k) -= 0.5*H*div;
    break;

    //
    // two gas neighbors (adjacent)
    //

case NF_101000:
    U(i-1,j,k)=U(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j-1,k)=V(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_100100:
    U(i-1,j,k)=U(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j,k)=V(i,j-1,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_011000:
    U(i,j,k)=U(i-1,j,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j-1,k)=V(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_010100:
    U(i,j,k)=U(i-1,j,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j,k)=V(i,j-1,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;

```

```

case NF_001010:
    V(i,j-1,k)=V(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_001001:
    V(i,j-1,k)=V(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_000110:
    V(i,j,k)=V(i,j-1,k)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_000101:
    V(i,j,k)=V(i,j-1,k)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;

case NF_100010:
    U(i-1,j,k)=U(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_100001:
    U(i-1,j,k)=U(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_010010:
    U(i,j,k)=U(i-1,j,k)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_010001:
    U(i,j,k)=U(i-1,j,k)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;

    //
    // three gas neighbors (adjacent/liquid corner)
    //

case NF_010101:
    U(i,j,k) = U(i-1,j,k);

```



```

        V(i,j,k) = V(i,j-1,k);
        W(i,j,k) = W(i,j,k-1);
        break;
case NF_100101:
    U(i-1,j,k) = U(i,j,k);
    V(i,j,k) = V(i,j-1,k);
    W(i,j,k) = W(i,j,k-1);
    break;
case NF_011001:
    U(i,j,k) = U(i-1,j,k);
    V(i,j-1,k) = V(i,j,k);
    W(i,j,k) = W(i,j,k-1);
    break;
case NF_101001:
    U(i-1,j,k) = U(i,j,k);
    V(i,j-1,k) = V(i,j,k);
    W(i,j,k) = W(i,j,k-1);
    break;
case NF_010110:
    U(i,j,k) = U(i-1,j,k);
    V(i,j,k) = V(i,j-1,k);
    W(i,j,k-1) = W(i,j,k);
    break;
case NF_100110:
    U(i-1,j,k) = U(i,j,k);
    V(i,j,k) = V(i,j-1,k);
    W(i,j,k-1) = W(i,j,k);
    break;
case NF_011010:
    U(i,j,k) = U(i-1,j,k);
    V(i,j-1,k) = V(i,j,k);
    W(i,j,k-1) = W(i,j,k);
    break;
case NF_101010:
    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);
    break;

//
// three gas neighbors (two opposite)

```

//

case NF_111000:

U(i-1,j,k) += Gx*DT;

U(i,j,k) += Gx*DT;

V(i,j-1,k) = V(i,j,k) + H*(
(U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);

break;

case NF_110100:

U(i-1,j,k) += Gx*DT;

U(i,j,k) += Gx*DT;

V(i,j,k) = V(i,j-1,k) - H*(
(U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);

break;

case NF_110010:

U(i-1,j,k) += Gx*DT;

U(i,j,k) += Gx*DT;

W(i,j,k-1) = W(i,j,k) + H*(
(U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);

break;

case NF_110001:

U(i-1,j,k) += Gx*DT;

U(i,j,k) += Gx*DT;

W(i,j,k) = W(i,j,k-1) - H*(
(U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);

break;

case NF_101100:

V(i,j-1,k) += Gy*DT;

V(i,j,k) += Gy*DT;

U(i-1,j,k) = U(i,j,k) + H*(
(V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);

break;

case NF_011100:

V(i,j-1,k) += Gy*DT;

V(i,j,k) += Gy*DT;

U(i,j,k) = U(i-1,j,k) - H*(
(V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);

break;

case NF_001110:

V(i,j-1,k) += Gy*DT;

V(i,j,k) += Gy*DT;

```

        W(i,j,k-1) = W(i,j,k) + H*(
            (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
        break;
case NF_001101:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    W(i,j,k) = W(i,j,k-1) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
    break;

case NF_100011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i-1,j,k) = U(i,j,k) + H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_010011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i,j,k) = U(i-1,j,k) - H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_001011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    V(i,j-1,k) = V(i,j,k) + H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_000111:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    V(i,j,k) = V(i,j-1,k) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;

//
// four gas neighbors (two opposite pairs)
//

```

```

case NF_111100:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
          (V(i,j,k)-V(i,j-1,k))/H +
          (W(i,j,k)-W(i,j,k-1))/H;
    U(i-1,j,k) += 0.25*H*div;
    U(i,j,k) -= 0.25*H*div;
    V(i,j-1,k) += 0.25*H*div;
    V(i,j,k) -= 0.25*H*div;
    break;
case NF_110011:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
          (V(i,j,k)-V(i,j-1,k))/H +
          (W(i,j,k)-W(i,j,k-1))/H;
    U(i-1,j,k) += 0.25*H*div;
    U(i,j,k) -= 0.25*H*div;
    W(i,j,k-1) += 0.25*H*div;
    W(i,j,k) -= 0.25*H*div;
    break;
case NF_001111:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    div = (U(i,j,k)-U(i-1,j,k))/H +
          (V(i,j,k)-V(i,j-1,k))/H +
          (W(i,j,k)-W(i,j,k-1))/H;
    V(i,j-1,k) += 0.25*H*div;
    V(i,j,k) -= 0.25*H*div;
    W(i,j,k-1) += 0.25*H*div;
    W(i,j,k) -= 0.25*H*div;
    break;

```

```
//
```

```

// four gas neighbors (one opposite pair)
//

case NF_110101:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j,k)=V(i,j-1,k)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_110110:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j,k)=V(i,j-1,k)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_111001:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k)=V(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_111010:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k)=V(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((U(i,j,k)-U(i-1,j,k))/H);
    break;
case NF_011101:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    U(i,j,k)=U(i-1,j,k)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k)=W(i,j,k-1)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_011110:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    U(i,j,k)=U(i-1,j,k)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_101101:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;

```

```

        U(i-1,j,k)=U(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
        W(i,j,k)=W(i,j,k-1)-0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
        break;
case NF_101110:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    U(i-1,j,k)=U(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    W(i,j,k-1)=W(i,j,k)+0.5*H*((V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_010111:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i,j,k)=U(i-1,j,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j,k)=V(i,j-1,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_011011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i,j,k)=U(i-1,j,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j-1,k)=V(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_100111:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i-1,j,k)=U(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j,k)=V(i,j-1,k)-0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_101011:
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i-1,j,k)=U(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    V(i,j-1,k)=V(i,j,k)+0.5*H*((W(i,j,k)-W(i,j,k-1))/H);
    break;

    //
    // five gas neighbors
    //

case NF_011111:
    V(i,j-1,k) += Gy*DT;

```

```

    V(i,j,k) += Gy*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i,j,k) = U(i-1,j,k) - H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_101111:
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    U(i-1,j,k) = U(i,j,k) + H*(
        (V(i,j,k)-V(i,j-1,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_110111:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    V(i,j,k) = V(i,j-1,k) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_111011:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
    V(i,j-1,k) = V(i,j,k) + H*(
        (U(i,j,k)-U(i-1,j,k))/H + (W(i,j,k)-W(i,j,k-1))/H);
    break;
case NF_111101:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    W(i,j,k) = W(i,j,k-1) - H*(
        (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
    break;
case NF_111110:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k) += Gy*DT;

```

```

V(i,j,k) += Gy*DT;
W(i,j,k-1) = W(i,j,k) + H*(
    (U(i,j,k)-U(i-1,j,k))/H + (V(i,j,k)-V(i,j-1,k))/H);
break;

//
// six gas neighbors
//

case NF_111111:
    U(i-1,j,k) += Gx*DT;
    U(i,j,k) += Gx*DT;
    V(i,j-1,k) += Gy*DT;
    V(i,j,k) += Gy*DT;
    W(i,j,k-1) += Gz*DT;
    W(i,j,k) += Gz*DT;
break;

```


Index

- absorption coefficient, 118
- albedo, 119
- boundary cells, 50
- BRDF, 118
- BSDF, 118
- BTDF, 118
- clonal mosaic model, 21
- combustion, 15
- compressible flow, 36
- computational cells, 51
- conjugate gradient method, 62
- continuity equation, 40–41, 53
- Courant-Friedrichs-Levy condition, 64
- deformation tensor, 41
- dynamic viscosity, 34
- energy equation, 42–43, 54
- erosion, 19
- Euler, Leonhard, 31
- Eulerian, 35
- explosion, 14
- finite differences, 47–49
- fire, 15
- flow field, 10
- fluid
 - definition, 32–33
- Fourier synthesis, 11
- free-slip boundary, 58
- gas cells, 51
- height field, 11
- Henvey-Greenstein model, 126
- implicit surface, 10
- incompressible flow, 36
- kinematic viscosity, 34
- Lagrangian, 35
- liquid cells, 51
- Mach number, 36
- marker-and-cell method, 12

material derivative, 35
 Mie scattering, 125
 momentum equation, 41–42, 54
 Navier, Claude, 31
 neighboring cells, 51
 Newtonian fluid, 38
 no-slip boundary, 58
 outflow boundary, 58
 particle systems, 10
 periodic boundary, 58
 phase function, 119, 123–126
 photon tracing, 126–129
 radiance equation, 120
 ray tracing, 129–131
 Rayleigh scattering, 125
 reaction-diffusion, 20
 reactive elements, 27–29
 scattering coefficient, 118
 semi-Lagrangian integration, 13, 49
 smoothed particle hydrodynamics, 10
 solid cells, 51
 staggered grid, 51
 Stokes derivative, 35
 Stokes, George, 31
 Strauss shading model, 122
 stress tensor, 41
 successive over-relaxation method, 62
 terrain modelling, 19
 texture-mapping, 17
 transmittance, 119
 transport theorem, 40
 wave models, 9
 weathering, 17–19