PHYSICS-BASED MODELING OF GASEOUS PHENOMENA FOR COMPUTER GRAPHICS

By
J. G. Mason

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER SCIENCE
AT
DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA
AUGUST 13, 2004

DALHOUSIE UNIVERSITY

DEPARTMENT OF

COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Computer Science for acceptance a thesis entitled "**Physics-based Modeling of Gaseous Phenomena for Computer Graphics**" by **J. G. Mason** in partial fulfillment of the requirements for the degree of **Master of Computer Science**.

Dated: August 13, 2004

Supervisors: ____________________

Dr. J. Norman Scrimger

____

Dr. Raymond J. Spiteri

Reader: ____________________

Dr. Dirk V. Arnold

DALHOUSIE UNIVERSITY

Date: **August 13, 2004**

Author: **J. G. Mason**

Title: **Physics-based Modeling of Gaseous Phenomena for Computer Graphics**

Department: **Computer Science**

Degree: **MCSc** Convocation: **October** Year: **2004**

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

Table of Contents

List of Tables vi

List of Figures vii

Acknowledgements ix

Abstract x

Chapter 1 Introduction 1

1.1 Background 1

1.2 Literature Survey 4

1.2.1 Previous work 4

1.2.2 Current Work 5

Chapter 2 The Method of Foster and Metaxas 8

2.1 Introduction 8

2.2 The Navier-Stokes Equations for Incompressible Fluid Flow with Thermal Buoyancy 10

2.2.1 Temperature Distribution 11

2.2.2 Conservation of Mass 12

2.3 Discretization of the Acceleration Equation 13

2.4 Discretization of Temperature Distribution 20

2.5 Discretization of Incompressibility Condition 21

2.6 The Turbulent Gas Algorithm 22

Chapter 3 Method of Foster and Metaxas: Critique and Improvements 24

3.1 Introduction 24

Chapter 4 Implementation and Results 33

4.1 Steam room animation 35

4.2 Smokestack animation 35

4.3 Object Interaction animation 36

4.4 Discussion 37

Chapter 5 Conclusion 47

5.1 Future Work 49

Bibliography 52

List of Tables

4.1 Specific parameters used to produce each animation. 35

4.2 Profiling of the Steam Room animation as recorded in seconds. 39

4.3 Profiling of the Smokestack animation as recorded in seconds. 40

4.4 Profiling the object interaction animation as recorded in seconds. . . 45

4.5 Mass convergence coefficients for the Steam Room, Smokestack, and FCS animations. 46

List of Figures

2.1 With the grid defined, each voxel has a temperature and pressure at its centre and velocities on its faces. 10

3.1 A flow moving diagonal to the grid. Diagonal flows tend to exhibit an exaggerated diffusion. 27

3.2 Flow moving parallel to the grid. 28

3.3 A flow moving along both a smooth and jagged surface (grid resolution 11×11 and particle size 0.2). 29

3.4 A flow moving along both a smooth and jagged surface (grid resolution 22×22 and particle size 0.2). 30

3.5 A flow moving along both a smooth and jagged surface (grid resolution 11×11 and particle size 0.4). 31

3.6 A flow moving along both a smooth and jagged surface (grid resolution 33×33 and particle size 0.4). 31

3.7 Schematic diagram of Buck's method shown here for a two-dimensional grid. 32

4.1 The orange framework depicts the voxels that comprise the steam nozzles. 36

4.2 Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 150. 37

4.3 Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 325. 38

4.4 Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 500. 39
4.5 The orange framework depicts the voxels that comprise the smokestack. 40
4.6 Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 60. 41
4.7 Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 80. 42
4.8 Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 100. 43
4.9 The orange framework depicts the voxels that comprise the letters FCS. 43
4.10 Smoke drifting around solid objects in the form of the letters FCS at Frame 20. 44
4.11 Smoke drifting around solid objects in the form of the letters FCS at Frame 50. 44
4.12 Smoke drifting around solid objects in the form of the letters FCS at Frame 150. 45

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. J. Norman Scrimger, whose intelligence, guidance, understanding, and patience throughout the course of my entire university education has not only served to encourage me to successfully realize my goals, but whose mentoring has also inspired me to be a better person.

I would also like to thank my co-supervisor, Dr. Raymond J. Spiteri. The thesis would certainly not have been possible without his vast expertise as a mathematician, but just as important is his ability to instill that knowledge upon others, thus transforming the highly complex into the comprehendible.

A very special thank you goes to Peter F. Spierenburg, who over the years has unselfishly helped me and many others to understand topic after topic in computer science thanks to his vast knowledge of the discipline.

Finally, and most importantly, I would like to dedicate this thesis to my parents, Gerald H. Mason and D. Anne Mason, whose support and encouragement has made this accomplishment possible. I am truly blessed to have such wonderful and loving parents.

Abstract

Physics-based modeling plays a crucial role in many industrial and scientific sectors. Industries such as aerospace, manufacturing, wireless communication, and resource recovery all rely heavily on physics-based modeling for design and decision support. The power and popularity of physics-based modeling is also proliferating into new and exciting areas such as film-making, computer games, and virtual reality. Moreover, physics-based modeling is now generally accepted as a third principal mode of scientific investigation, along with theory and experiment.

Physically accurate results are obviously of paramount importance in scientific and engineering fields, and the same holds true for the entertainment industry. The difference is that for entertainment purposes, it is often more important to demonstrate a look and feel that is convincing to an audience rather than to focus on high accuracy. Moreover, rendering time costs money, and many consumer products such as three-dimensional modeling and video games often have real-time requirements. For these reasons, the prospect of fast simulation methods that yield believable results is very attractive to the entertainment industry. The role of the computer scientist in this endeavor is crucial, but more often than not, knowledge is achieved through an interdisciplinary approach involving mathematicians and engineers as well.

This thesis discusses physics-based modeling of gaseous phenomena for computer animation. Specifically the focus is on approximating the solution to the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy in order to achieve automated and believable special effects. This includes a detailed overview and discussion of related literature and an examination of the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy. We also give detailed analysis of the explicit finite difference approximation method of Foster and Metaxas [7] and suggest some improvements that reduce the computation time of their mass conservation routine, resulting in faster rendering. These improvements include the application of a generalization of Buck's method [13] to three dimensions.

Additionally, several special effects conceived by Foster and Metaxas [7] have been reproduced. These include an animation of hot steam as it is released from a valve in a steam room, smoke leaving a smokestack in a wind-filled environment, and a solid object interaction demonstrating smoke flowing around the three block letters FCS (Faculty of Computer Science).

Chapter 1

Introduction

1.1 Background

Scientists and engineers have been using physics to model the complex behaviour of gaseous phenomena such as smoke and steam for decades. For example, physics-based modeling is prevalent in several fields of science and engineering, having extensive application to such areas as aerospace, manufacturing, wireless communication, and resource recovery. More recently, physics-based modeling has been shown to have an intriguing application to the entertainment industry, yielding believable special effects for both movies and television, as well for virtual environments such as computer gaming.

Speaking in terms of a general application to science and engineering, physics-based modeling has proven to be invaluable as a means of predicting the real-world behaviour of everything from existing natural phenomena to man-made products still in preproduction mode. The ability to conduct simulations in a physically accurate virtual world lends itself to savings in time and money, all with the added benefit of eliminating the potential hazards that may exist when conducting experiments in the real world. Of course where scientific and engineering applications are concerned, *accuracy* is always a crucial issue. The degree of required accuracy varies depending upon the specific application, but in general, a high level of accuracy is assumed for scientific and engineering simulations. This is in great contrast to physics-based modeling as applied to the entertainment industry, where *believability* is the only goal, and the real world adaptation is of no importance. Realizing that a believable simulation does not require the same level of accuracy as a scientific or engineering application allows researchers to search for ways of speeding up the animation process by approximating the required physics as opposed to finding the highly accurate results typically required by other industries.

Although it is true that this approach places less emphasis upon high accuracy (because the simulation need only look convincing), it is by no means an implication that mathematics or physics are not important; in fact quite the opposite is true as they both have a pivotal role to play. By applying physics-based modeling techniques to computer graphics we can attempt to model and visualize phenomena in a believable fashion based upon physically correct motions and interactions with the environment. Basing special effects upon approximated physics keeps the animation looking natural, while vastly reducing the computation time required to render each frame. Using physics also helps to automate the animation process, resulting in the reduction of valuable human design time. With audiences constantly demanding more realism, the days when an animation artist could create believable effects without the aid of physics-based modeling are certainly numbered. Even for less complex simulations where strict accuracy is not important, mathematics and physics can still play an important role.

The idea here is certainly not to replace the creativity of a human animator with a computer, but rather to empower the animator by providing animation tools that afford a high level of control to the user, while automating the entire process in an efficient manner. In the case of modeling the motion of fluid-like effects such as smoke or steam, the simulation is often based upon a solution of the Navier-Stokes equations for incompressible fluid flow, a set of partial differential equations that describe the motion of fluid as it evolves temporally through a three-dimensional domain.

This thesis discusses physics-based modeling of gaseous phenomena for computer animation, with a focus on approximating the solution to the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy in order to achieve automated and believable special effects. The remainder of this chapter covers previous fluid motion research concerning the behaviour of gaseous phenomena. Chapter 2 presents an introduction to the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy, including an in-depth analysis of the explicit finite difference approximation method developed by Foster and Metaxas [7]. Chapter 3 provides a critique of the method of Foster and Metaxas, with suggested improvements to their method. Chapter 4 demonstrates the results generated from the improvements of chapter 3. Finally, chapter 5 discusses conclusions and directions for future research.

1.2 Literature Survey

1.2.1 Previous work

William T. Reeves was perhaps first to demonstrate the usefulness of computer-generated animation when his special effects wowed audiences in the 1983 science fiction film *Star Trek II: The Wrath of Khan* [1]. In a ground-breaking paper [4], Reeves introduced a computer automation technique centred around a stochastically controlled particle system which he used to generate what he coined as *fuzzy objects*. These were phenomena such as fire, clouds, and water; objects that did not have distinct edges. Rendering objects using particle systems rather than the more traditional surfaced-based modeling techniques such as static bitmaps lead to a more natural appearance than those produced by more traditional polygonal geometry. In 1985 Reeves published a second paper [5] where he introduced *approximate* and *probabilistic algorithms* developed to provide more believable shading for his fuzzy objects. He also furthered his stochastic modeling technique to animate trees and grass so as to exhibit complex motions such as swaying in wind. This technique was exhibited in the short film *The Adventures of Andre and Wally B* [2].

The main goal of Reeves' research was to generate believable animations for movie-going audiences, and although his work was not physically accurate, it was certainly ground-breaking. Mathematical equations (such as the Navier-Stokes equations) capable of describing complex physical motion were certainly well known at the time, but it is worthwhile considering that given the available computing power of the day; solving complex equations in three dimensions was not likely to be a practical approach. In any case, the special effects produced from his research did render visually believable effects, having passed the scrutiny of thousands of movie goers world-wide.

Despite the success of Reeves' stochastically based animation methods, there were still serious drawbacks. Because his particle systems acted independently, Reeves' stochastic modeling schemes were incapable of interacting with the surrounding medium. This meant that the look and feel of the animation was left almost entirely in human hands, leaving the animator with the time-consuming task of separately addressing areas where overlapping systems would require the *appearance* of interacting with each other. Because so much of the work is left up to the animator to create a scene, results could be unpredictable, potentially ending with animations having an unrealistic feel [7]. Requiring the animator to have this level of *micro-control* over parameters could still work fairly well for relatively small-scale projects; however if the complexity of the project continued to grow, it could eventually reach a point where it would be no longer practical for even the most talented of animation artists [7].

1.2.2 Current Work

Since Reeves' work in the 1980's, there has been a significant amount of research revolving around increasing the believability of simulations for computer graphics. Recent work has focused on using physics-based modeling to produce more believable animations, while also lessening the dependence on the animator by largely automating the animation process. Physics-based simulations have the inherent advantage of allowing complex phenomena such as smoke or steam to interact with the surrounding environment. This interaction is automatic and looks natural because both object and environment data are accounted for in the equations that model the motion. This physics-based approach empowers the animator by giving general control over global activities of a scene without requiring concern for *tweaking* localized areas where systems may converge or collide.

An excellent introductory example of solving equations for automated fluid flow is demonstrated in [7]. This method is based largely upon previous work published by Harlow and Welch in 1965 [3]. The approach of Foster and Metaxas was to solve the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy by using an explicit finite difference method. This method was shown to work well for rendering the motion of hot turbulent gases, provided that the time steps upon which the attributes were updated did not allow for flow movements greater than the length of one discrete cell per update. Attempting to model faster flows would result in an unstable solution, ruining the animation.

Stam [8] addressed the problem of stability by proposing an unconditionally stable method for solving the Navier-Stokes equations. Before this, methods such as that of Foster and Metaxas [7] had employed a strictly Eulerian approach, in which equations were solved explicitly on a fixed grid. This new method by Stam used a semi-Lagrangian approach, solving the equations implicitly using a grid in combination with a particle tracking scheme and invoking implicit methods to solve the Navier-Stokes equations. Stam's approach was not only more stable, but it was also faster due to the allowance of larger time steps [8, 9]. Unfortunately the method suffered from a premature dampening of flow or *numerical dissipation*, meaning that the flow tended to slow to a stop earlier than a real fluid. The final product still had an overall motion that was fluid-like; however Stam reported the occurrence of numerous tiny vortices which gave the appearance of smoke disappearing at an unnaturally fast rate [8]. This is why semi-Lagrangian schemes are seldom used in engineering-related simulations; however for non-scientific applications in computer graphics this makes little difference to believability when modeling gaseous phenomena such as gas or smoke [9, 11].

In a recently published method that does not rely on Navier-Stokes equations, Fedkiw, Stam, and Jensen [10] used a semi-Lagrangian method much the same as Stam's [8] to solve for the *incompressible Euler equations*, which they claim to be "usually more appropriate for gas modeling and less computationally expensive than the viscous Navier-Stokes equations". Their scheme ensures stability of the model even at large time steps. It also solves Stam's [6] problem of numerical dissipation by borrowing from the field of computational fluid dynamics and using a "vorticity confinement", which is basically a scheme for reintroducing the energy lost due to the numerical dissipation. This lost energy is injected back into the fluid field by using a *forcing term* that increases the vorticity of the flow, keeping the otherwise dissipating flow in motion [10].

Chapter 2

The Method of Foster and Metaxas

2.1 Introduction

For over a decade the vast majority of fluid animation research for computer graphics has been based upon solving a set of partial differential equations known as the Navier-Stokes equations for incompressible fluid flow. Modeling complex behaviour of phenomena such as smoke or steam on a molecular scale would be an extremely complex process [14] and would certainly make for unreasonable computation times. Instead, the Navier-Stokes equations for incompressible fluid flow calculate the behaviour of fluid by considering the medium as comprised of a group of incompressible volumes of molecules [14]. Provided that these volumes are chosen small enough to ensure negligibility in pressure, velocity, and/or temperature changes inside themselves [14], the Navier-Stokes equations are capable of accurately describing the physical behaviour of a fluid as it evolves over space and time as a whole [14, 8].

One of the earliest utilizations of the Navier-Stokes equations for the production of special effects is described in [7], wherein the Navier-Stokes equations are used to develop an automated animation method that is capable of rendering convincing effects of smoke and steam. This method discretizes the animation area into a series of cubic cells called volume elements or *voxels*, thereby reducing the complexity of the animation area. This process also forms a grid upon which the velocity, temperature, and pressure gradient are defined (see Figure 2.1). The centre of each voxel is identified by its i, j, k position relative to the x, y, z coordinates of the grid. Using this system, the average temperature and pressure are defined at the centre of each voxel as $T_{i,j,k}$ and $p_{i,j,k}$ respectively. Velocities are defined on voxel faces as u, v, w, and are also denoted with respect to the i, j, k system, allowing for the velocity at $i+1/2, j, k$ to be written as $U_{i+1/2,j,k}$. Also, because each voxel shares its faces with the faces of immediate neighbours, velocities defined on faces are shared between voxels. For instance, the velocity $U_{i+1/2,j,k}$ on a face defined at voxel i, j, k is shared with the neighbouring voxel $i+1, j, k$. The velocity at a voxel's centre i, j, k is found by averaging the velocities of the opposing walls. For example $U_{i,j,k} = (U_{i-1/2,j,k} + U_{i+1/2,j,k})/2$. In general, velocities at any point in a voxel are approximated by linearly interpolating the values on the faces.

Utilizing the grid, the Navier-Stokes equations are discretized by applying a finite difference method. It is then possible to visualize the motion of this otherwise invisible gas by utilizing the *marker-and-cell* method of Harlow and Welsh [3], whereby massless particles are launched into the animation area. These particles are animated by voxel velocities and have no influence on the motion of the gas.

The remainder of this chapter provides a closer examination of the method described by Foster and Metaxas [7], including a detailed account of the equations and an overview of their “Turbulent Gas Algorithm”.

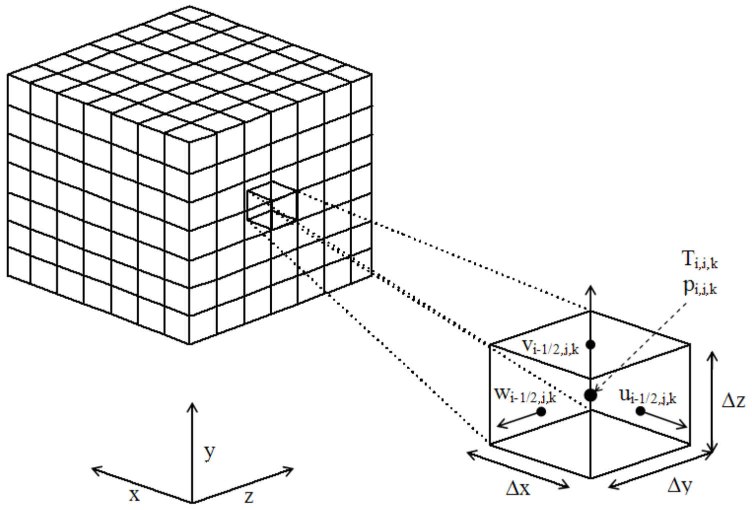

Figure 2.1: With the grid defined, each voxel has a temperature and pressure at its centre and velocities on its faces.

2.2 The Navier-Stokes Equations for Incompressible Fluid Flow with Thermal Buoyancy

The Navier-Stokes equations for incompressible fluid flow with thermal buoyancy model fluid motion by including the dominant physical traits which govern the motion of a fluid, namely, diffusion $\nu(\nabla \cdot \nabla)\mathbf{u}$, convection $(\mathbf{u} \cdot \nabla)\mathbf{u}$, pressure gradient ∇p, and the force of thermal buoyancy $\mathbf{F}$. Although these equations do not model gaseous phenomena on a molecular scale, they do present a good platform for predicting global movement. The equations take the form:

$$\frac{\partial \mathbf{u}}{\partial t} = \nu(\nabla \cdot \nabla)\mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p + \mathbf{F}, \tag{2.1}$$

where ∇ is the gradient operator, $\cdot$ is the inner product operator, ν is viscosity, and $\mathbf{u} = (u, v, w)$ is the velocity vector field of the fluid at each point in the domain.

2.2.1 Temperature Distribution

The force of thermal buoyancy requires a model of temperature distribution as it evolves throughout the grid. This is calculated by the following partial differential equations for temperature distribution:

$$\frac{\partial T}{\partial t} = \lambda(\nabla \cdot \nabla)T - \nabla \cdot (T\mathbf{u}), \tag{2.2}$$

where λ is the molecular diffusion constant, $\lambda(\nabla \cdot \nabla)T$ represents diffusion, and $\nabla \cdot (T\mathbf{u})$ represents convection.

The evolution of temperature within a voxel is assumed to be mainly dictated by the processes of diffusion and convection caused by energy exchanges with surrounding voxels.

2.2.2 Conservation of Mass

The algorithm in [7] gains its speed primarily because the Navier-Stokes equations are solved using an explicit finite difference method over a relatively coarse grid of voxels. Although this method has the advantage of speed, it also has the unfortunate side effect of suffering from large numerical error, which mainly manifests itself in individual voxels by altering the overall system mass.

In order to correct for these miscalculations of mass, the Navier-Stokes equations are appended with an incompressibility condition

$$0 = \nabla \cdot \mathbf{u}, \tag{2.3}$$

which insists that all flow entering a given voxel must be equal to the flow exiting. Equation (2.3) is enforced at the centre of each voxel. This ensures the preservation of mass throughout the grid, allowing the animation to exhibit the natural swirling motions often created when fluid is forced to flow back on itself by such influences as barriers or other flows.

2.3 Discretization of the Acceleration Equation

Discretization of the Navier-Stokes equations is achieved by applying the following finite difference approximation:

$$u^{n+1}_{i+1/2,j,k} =$$

$$u^{n}_{i+1/2,j,k} + \Delta t[\nu(\nabla \cdot \nabla)u^{n}_{i+1/2,j,k} - (u^{n}_{i+1/2,j,k} \cdot \nabla)u^{n}_{i+1/2,j,k} - \nabla p^{n}_{i+1/2,j,k} + F^{n}_{i+1/2,i,k}] \tag{2.4}$$

$$v^{n+1}_{i,j+1/2,k} =$$

$$v^{n}_{i,j+1/2,k} + \Delta t[\nu(\nabla \cdot \nabla)v^{n}_{i,j+1/2,k} - (v^{n}_{i+1/2,j,k} \cdot \nabla)v^{n}_{i,j+1/2,k} - \nabla p^{n}_{i,j+1/2,k} + F^{n}_{i,i+1/2,k}] \tag{2.5}$$

$$w^{n+1}_{i,j,k+1/2} =$$

$$w^{n}_{i,j,k+1/2} + \Delta t[\nu(\nabla \cdot \nabla)w^{w}_{i,j,k+1/2} - (w^{n}_{i+1/2,j,k} \cdot \nabla)w^{n}_{i,j,k+1/2} - \nabla p^{n}_{i,j,k+1/2} + F^{n}_{i,i,k+1/2}] \tag{2.6}$$

The subscripts i, j, k indicate the discrete location of the quantity in space, and the superscript n indicates the time step. For example, $u^{n}_{i+1/2,j,k} \approx u((i+1/2)\Delta x,\ j\Delta y,\ k\Delta z,\ n\Delta t)$ where Δx, Δy, and Δz are the voxel dimensions, and Δt is the time step size (refer to Figure 2.1).

A detailed account of the diffusion, convection, pressure gradient, and thermal buoyancy components follows.

Diffusion $\nu(\nabla \cdot \nabla)\mathbf{u}$:

Diffusion is the process by which energy contained in a voxel is spontaneously distributed to immediate neighbours of lower thermal concentration, effectively causing drag. For the u velocity component, the discretization results in the following update of velocity in the x direction:

$$\begin{aligned} u^{n+1}_{i+1/2,j,k} \quad = \quad & u^n_{i+1/2,j,k} + \nu\Delta t \Bigg[\frac{u^n_{i+3/2,j,k} - 2u^n_{i+1/2,j,k} + u^n_{i-1/2,j,k}}{(\Delta x)^2} + \\ & \frac{u^n_{i+1/2,j+1,k} - 2u^n_{i+1/2,j,k} + u^n_{i+1/2,j-1,k}}{(\Delta y)^2} + \\ & \frac{u^n_{i+1/2,j,k+1} - 2u^n_{i+1/2,j,k} + u^n_{i+1/2,j,k-1}}{(\Delta z)^2} \Bigg], \end{aligned} \tag{2.7}$$

where ν is the coefficient of kinematic viscosity. Smaller values of ν imply lower viscosity or *drag*, leading to relatively higher convective and rotational velocity. The same methodology is used to update the v and w velocity components.

Convection $(\mathbf{u} \cdot \nabla)\mathbf{u}$:

As with diffusion, convection also represents a transfer of energy between neighbouring voxels. However unlike diffusion, energy transfer is directional, occurring due to motion created by relatively hot or cold voxels in the grid. In order to facilitate further analysis, we write the convection terms more explicitly,

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = \left[(u, v, w) \cdot \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)\right](u, v, w) \tag{2.8}$$

$$= \left(u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z}\right)(u, v, w) \equiv (\text{component 1, component 2, component 3}). \tag{2.9}$$

Component 1

Writing component 1 out explicitly we have:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}. \tag{2.10}$$

Because the incompressibility condition (2.3) in expanded form is

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{2.11}$$

we add

$$0 = u\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \tag{2.12}$$

to (2.10) and use the product rule of differentiation to write

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = \frac{\partial}{\partial x}(u^2) + \frac{\partial}{\partial y}(uv) + \frac{\partial}{\partial z}(uw). \tag{2.13}$$

Thus we discretize the first component of the convection term as

$$[(\mathbf{u}\cdot\nabla)\mathbf{u}^n_{i+1/2,j,k}]_1 = \left[\frac{(u^2)_{i+1,j,k}-(u^2)_{i,j,k}}{\Delta x}+\frac{(uv)_{i+1/2,j+1/2,k}-(uv)_{i+1/2,j-1/2,k}}{\Delta y}+\frac{(uw)_{i+1/2,j,k+1/2}-(uw)_{i+1/2,j,k-1/2}}{\Delta z}\right] \tag{2.14}$$

Components 2 and 3 are treated in a similar manner.

Only face velocities are stored in memory directly, so intermediate values for terms such as $(u^2)_{i+1,j,k}$ and $(u^2)_{i,j,k}$, are found by averaging the nearest face values, e.g.,

$$(u^2)_{i,j,k} = \frac{(u^2)_{i+1/2,j,k}+(u^2)_{i-1/2,j,k}}{2}. \tag{2.15}$$

The terms involving uv and uw are treated similarly.

Pressure Gradient ∇p:

Based upon an adaptation of the method of Harlow and Welch [3], the method of Foster and Metaxas [7] handles the pressure gradient by creating a separate field denoted by Ψ. The Ψ field, which is calculated by iterating over the grid using a classic three-dimensional Poisson equation solver, does not represent anything physical, but its gradient is equal to the pressure gradient. A Poisson equation takes the form:

$$(\nabla \cdot \nabla)\mathbf{u} = f(x, y, z), \tag{2.16}$$

where $f(x, y, z)$ is a prescribed function. Expanding (2.16) yields:

$$(\nabla \cdot \nabla)\mathbf{u} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f(x, y, z). \tag{2.17}$$

The potential field Ψ is defined at the centre of every voxel, and $\Psi^{(0)}_{i,j,k}$ is initially zero. The Ψ field is then updated by iterating over the field according to:

$$\begin{aligned} \Psi^{(s+1)}_{i,j,k} \quad = \quad & \frac{2}{\frac{8}{h^2}} \Bigg[-(\nabla \cdot \mathbf{u})_{i,j,k} + \frac{1}{h^2} \\ & \{\Psi^{(s)}_{i+1,j,k} + \Psi^{(s)}_{i-1,j,k} + \Psi^{(s)}_{i,j+1,k} + \Psi^{(s)}_{i,j-1,k} + \Psi^{(s)}_{i,j,k+1} + \Psi^{(s)}_{i,j,k-1}\} \Bigg] - \Psi^{(s)}_{i,j,k}, \end{aligned} \tag{2.18}$$

where $s = 0, 1, 2, ...$ is the iteration number, $(\nabla \cdot \mathbf{u})_{i,j,k}$ is mass divergence at the voxel centre i, j, k, and for simplicity we have written $\Delta x = \Delta y = \Delta z = h$. The iteration continues until convergence of the field is achieved as determined by:

$$\left| \frac{|\Psi^{(s+1)}_{i,j,k}| - |\Psi^{(s)}_{i,j,k}|}{|\Psi^{(s+1)}_{i,j,k}| + |\Psi^{(s)}_{i,j,k}|} \right| < \varepsilon, \tag{2.19}$$

where $\varepsilon = 10^{-4}$ (found by experimentation [7]). In other words, the iteration continues until the relative change in the Ψ field from one iteration to the next is less than 0.01%.

Once Ψ has been calculated for all voxels, the Ψ field represents the overall *relative* mass discrepancy from voxel to voxel, and because $\nabla\Psi$ is defined to be equal to ∇p [7], corrections to $\mathbf{u}$ can be made directly:

$$u^{n+1}_{i+1/2,j,k} \leftarrow u^{n+1}_{i+1/2,j,k} - \frac{\Psi_{i+1,j,k} - \Psi_{i,j,k}}{\Delta x},$$

$$v^{n+1}_{i,j+1/2,k} \leftarrow v^{n+1}_{i,j+1/2,k} - \frac{\Psi_{i,j+1,k} - \Psi_{i,j,k}}{\Delta y},$$

$$w^{n+1}_{i,j,k+1/2} \leftarrow w^{n+1}_{i,j,k+1/2} - \frac{\Psi_{i,j,k+1} - \Psi_{i,j,k}}{\Delta z}. \tag{2.22}$$

Thermal Buoyancy Force F:

Thermal buoyancy is the force that acts against gravity as fluid is convected in the y direction. We express this force as

$$\mathbf{F}_{i,j+1/2,k} = \begin{pmatrix} 0 \\ -\beta g(T_0 - T_{i,j+1/2,k}) \\ 0 \end{pmatrix}, \tag{2.23}$$

where β is the coefficient of thermal buoyancy, g is the acceleration due to gravity, T_0 is the ambient temperature, and $T_{i,j+1/2,k}$ is the average boundary temperature between $T_{i,j,k}$ and $T_{i,j+1,k}$. Thermal buoyancy is a direct result of temperature changes from within the fluid. Therefore, a model of temperature distribution across the grid must also be defined.

2.4 Discretization of Temperature Distribution

Discretization of the partial differential equation that governs diffusion and convection for temperature distribution is analogous to the discretization process for the velocities (2.1). This results in the following discretization:

$$T_{i,j,k}^{n+1} = T_{i,j,k}^{n} + \Delta t[\lambda(\nabla \cdot \nabla)T_{i,j,k}^{n} - \nabla \cdot (T\mathbf{u})_{i,j,k}^{n}]. \tag{2.24}$$

A detailed account of the diffusion and convection components follows.

Diffusion: $\lambda(\nabla \cdot \nabla)T$:

$$\begin{aligned} T_{i,j,k}^{n+1} \quad = \quad & T_{i,j,k}^{n} + \lambda\Delta t\left[\frac{T_{i+1,j,k}^{n} - 2T_{i,j,k} + T_{i-1,j,k}^{n}}{(\Delta x)^2} + \right. \\ & \frac{T_{i,j+1,k}^{n} - 2T_{i,j,k} + T_{i,j-1,k}^{n}}{(\Delta y)^2} + \\ & \left. \frac{T_{i,j,k+1}^{n} - 2T_{i,j,k} + T_{i,j,k-1}^{n}}{(\Delta z)^2}\right]. \end{aligned} \tag{2.25}$$

Convection $\nabla \cdot (T\mathbf{u})$:

$$\nabla \cdot (T\mathbf{u})_{i,j,k}^{n} = \left[\frac{\partial}{\partial x}(Tu) + \frac{\partial}{\partial y}(Tv) + \frac{\partial}{\partial z}(Tw)\right]_{i,j,k}^{n} \tag{2.26}$$

$$
\begin{aligned}
= \quad & \frac{(Tu)^n_{i+1/2,j,k} - (Tu)^n_{i-1/2,j,k}}{\Delta x} + \\
& \frac{(Tv)^n_{i,j+1/2,k} - (Tv)^n_{i,j-1/2,k}}{\Delta y} + \\
& \frac{(Tw)^n_{i,j,k+1/2} - (Tw)^n_{i,j,k-1/2}}{\Delta z},
\end{aligned}
\tag{2.27}
$$

where for example,

$$
(Tu)^n_{i+1/2,j,k} = u^n_{i+1/2,j,k} \frac{(T^n_{i+1,j,k} + T^n_{i,j,k})}{2}, \tag{2.28}
$$

$$
(Tu)^n_{i-1/2,j,k} = u^n_{i,j,k} \frac{(T^n_{i-1,j,k} + T^n_{i-1/2,j,k})}{2}. \tag{2.29}
$$

2.5 Discretization of Incompressibility Condition

An incompressibility condition is required to ensure the preservation of mass throughout the grid, allowing the animation to exhibit a natural motion.

The incompressibility condition (2.3) is discretized as follows:

$$
\begin{aligned}
(\nabla \cdot \mathbf{u})_{i,j,k} \quad = \quad & \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} + \\
& \frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta y} + \\
& \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta z} = 0.
\end{aligned}
\tag{2.30}
$$

2.6 The Turbulent Gas Algorithm

The design of the Turbulent Gas Algorithm [7] is composed of two parts. During Part 1 the user sets the appropriate environmental conditions necessary for producing the desired special effect. Once Part 1 has been completed, Part 2 proceeds to calculate the motion of the gas until the user stops the simulation.

Part 1

Step 1: Discretize the simulation area as a series of cubic cells (voxels) with side lengths of Δx, Δy, Δz.

Step 2: Select boundary conditions for any obstructive objects within the simulation area, as well as appropriate constant values for parameters such as viscosity (ν), thermal expansion (β), and molecular diffusion (λ).

Step 3: Choose a Δt that provides the desired frame rate while maintaining numerical stability. Stability is achieved by ensuring that the time-step Δt for a given gas velocity is less than the side length of a single voxel; i.e., $\Delta t\|\mathbf{u}\| < \min\{\Delta x, \Delta y, \Delta z\}$, where $\|\mathbf{u}\| = \sqrt{u^2 + v^2 + w^2}$.

Part 2

Step 4: The automatic part of the algorithm begins to run. The temperature and velocity for each voxel of the grid are solved for by using an explicit finite difference method to approximate (2.1) and (2.2).

Step 5: The mass divergence at the centre of each voxel is calculated using (2.30). To ensure conservation of mass, Ψ is iteratively calculated from (2.18) for every voxel in the grid until the convergence condition (2.19) is satisfied.

Step 6: Finally, the face velocities of each voxel are updated using (2.22), and the process starts over with the recalculation of the temperature and velocity at the centre of each voxel (step 4).

The algorithm continues until the animation sequence completes.

Chapter 3

Method of Foster and Metaxas: Critique and Improvements

3.1 Introduction

Foster and Metaxas [7] broke new ground when they presented a physics-based method for modeling the motion of a hot turbulent gas for three-dimensional computer animations. Their method, known as the Turbulent Gas Algorithm, was centred on discretizing the animation area to form a grid used to approximate a solution to the full three-dimensional Navier-Stokes equations for incompressible fluid flow. The results of their implementation successfully demonstrated that their method was capable of running faster than a method designed for strict accuracy, while still producing results that were believable [8]. In addition, their method had the advantage of being relatively straightforward to code, thanks to the explicit finite difference method used to solve the equations [7].

Simplifying the problem by discretizing the animation area was the key to the success of their algorithm; however it was also the cause of several shortcomings. For instance, their method is incapable of modeling fast-moving flows [17] that jump further than the side length of a single voxel in one time step [8, 10, 7]. This problem is not unworkable and can often be fixed simply by restarting the simulation with a smaller time step. Foster and Metaxas have since addressed this problem by setting a restriction on the time step; i.e., that the product of the maximum velocity for any voxel in the grid and the time step is always less than the minimum side length of the voxel [7].

Another problem associated with their discretized solution is that although a particle's position within a single voxel can be approximated through linear interpolation, rotational motion (i.e., directional change) cannot be solved for resolutions equal to or less than the side length of one voxel [7]. This relates to the trade-off made between accuracy and speed when using an explicit finite difference method. Foster and Metaxas suggest increasing the resolution of the entire grid until it produces visually acceptable results (at the expense of computation time) and suggest that this could be made more efficient by implementing an adaptive-grid solution whereby selected areas of particular interest could be solved for more accurately and at a lower overall cost than increasing the resolution of the entire grid [7]. However this is not trivial to code and does add to overhead in the form of increased computation time and memory usage.

Another concern, again relating to the discretized environment, is that of voxel orientation [7, 14]. One of the inherent problems of solving the Navier-Stokes equations over a finite grid is that results will vary according to the direction of the flow through the voxels. For example, a flow moving diagonally through the grid (see Figure 3.1) will tend to exhibit exaggerated turbulence. This is due to an increase in numerical diffusion as compared to that of flow moving parallel to the grid [7, 14] (see Figure 3.2). This exaggerated turbulence becomes even more of a concern when gas flow is interacting with walls. When the edge of a wall is oriented parallel to a grid axis, the flow will tend to move smoothly along the surface, carrying the marker particles with it. However, when a wall is oriented diagonally, the particles tend to get blocked, highlighting the jaggedness of the grid and making it difficult to create a believable-looking straight surface (see Figure 3.3). Again, the visual impact of this phenomenon can usually be lessened by increasing the resolution of the grid until the unwanted effect becomes less noticeable (see Figure 3.4), but as usual, this is at the expense of computation time. A less costly solution is to mask the look of the jagged edges by obscuring detail using larger marker particles (see Figures 3.5 and 3.6). This will help to soften the unwanted effect; however the effectiveness of this method is quite limited, and the overall buildup of the particles can look unnatural. Another possible solution is to try different shaped voxels; however this solution would be at a considerable increase in complexity and would result in a high computation time. Overall, it is best to align the voxel orientation so that the gas flow and/or object boundaries are parallel to an axis [7]. This is a considerable limitation, but it is a simple solution, and the flow will look more natural [7, 14].

For the most part these problems are not significantly restrictive to producing believable results. The resolution of the grid and the size of the time steps can usually be altered to provide believable results with computation times that are not unreasonable. In practice, the most serious problem is realized when attempting to implement the mass conservation method of Foster and Metaxas for maintaining stability of flow.

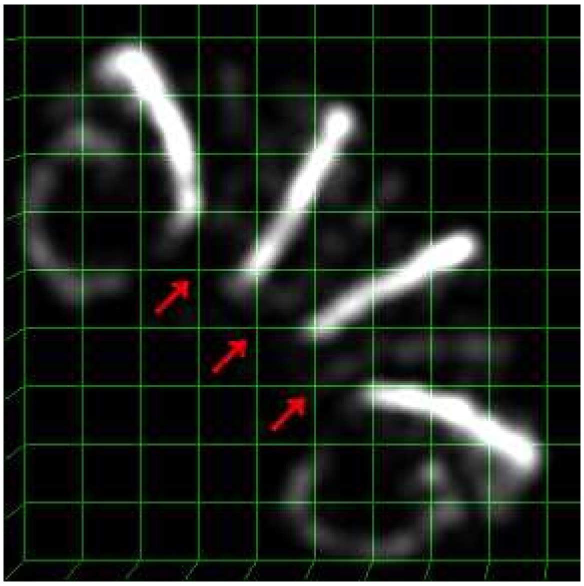

Figure 3.1: A flow moving diagonal to the grid. Diagonal flows tend to exhibit an exaggerated diffusion.

The Turbulent Gas Algorithm requires solving a classical three-dimensional Poisson equation based on a method developed by Harlow and Welsh with modifications made for use with three dimensions [3, 7]. As discussed in chapter 2, the problem with mass conservation is that the Ψ field can be very slow to converge, often suffering from stability problems [13] or not converging at all [14]. Foster [15], who states he could not pinpoint the exact cause, later posted an addendum suggesting a relaxation method designed to allow for less stringent convergence. Foster and Metaxas have since abandoned their Turbulent Gas Algorithm in favour of a slower but more dependable algorithm that utilizes the Navier-Stokes equations in combination with the Lagrange equations of motion. This new algorithm is more stable and always converges [16].

During the research phase of this thesis, the Turbulent Gas Algorithm [7] would

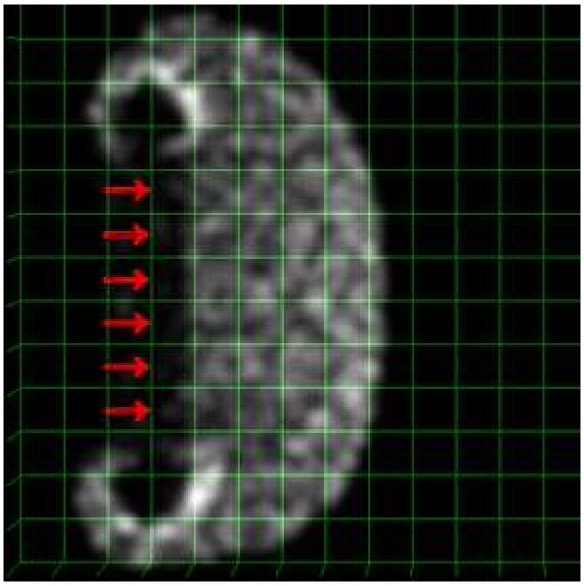

Figure 3.2: Flow moving parallel to the grid.

converge, but was either extremely slow or would not run stably. To speed up the algorithm, equation (2.16) was modified such that the Ψ field would begin iterating over the grid using the Ψ values from the previous iteration cycle, as opposed to starting with zero everywhere as in the non-modified algorithm. Although this did increase the speed of the algorithm to some degree, it can be shown that the stability of the algorithm is not changed; i.e., instability was still a problem. Departing from Foster and Metaxas algorithm altogether lead to a method developed by Buck [13] originally proposed in two dimensions that was adapted in this work for three dimensions. It was successful at producing believable and stable results with the additional benefit of decreased rendering time.

Buck's method works much the same way as the method of Foster and Metaxas; however through the course of this thesis it was also found to be simpler, faster, and more stable. The mass divergence is calculated for every voxel in the grid. In two dimensions, if the divergence for a given voxel is found to be greater than a preset

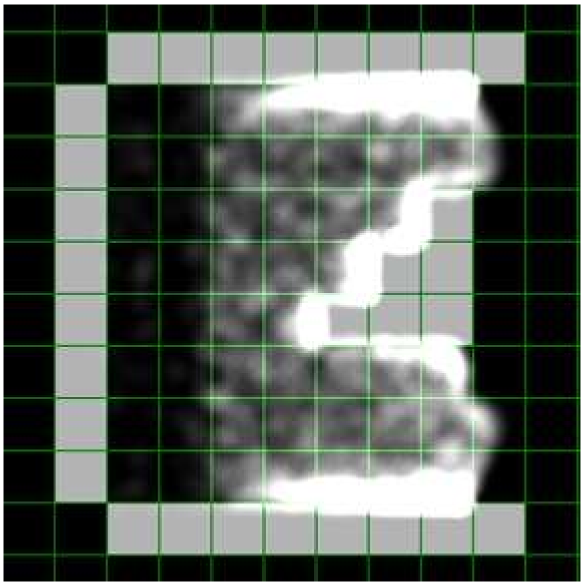

Figure 3.3: A flow moving along both a smooth and jagged surface (grid resolution 11×11 and particle size 0.2).

threshold, the divergence is moved toward zero by subtracting 1/4 of the voxel's total pressure gradient from the lagging face and adding 1/4 of the gradient to the leading face (see Figure 3.7). The algorithm iterates over the entire grid until the maximum mass divergence for any voxel in the grid is less than the preset threshold. We note that Buck gives no explanation or justification for using 1/4 to smooth the gradient; however it was assumed that 1/4 is used because in two-dimensional space, there are four neighbours to a voxel. Moreover, Buck likely found (but did not report) that this leads to the most efficient iteration. In this thesis, we generalize this assumption to three dimensions (as in Chapter 4); i.e., 1/6 is used instead of 1/4 because three-dimensional voxels have 6 neighbours. Working with values smaller than 1/6 yields a smoother distribution but at the cost of an increase in the required number of iterations and thus execution time. Using a larger value than 1/6 also causes an increase in execution time and may also compromise the stability of the algorithm. See Table 4.5 in chapter 4 for the number of iterations yielded by different thresholds

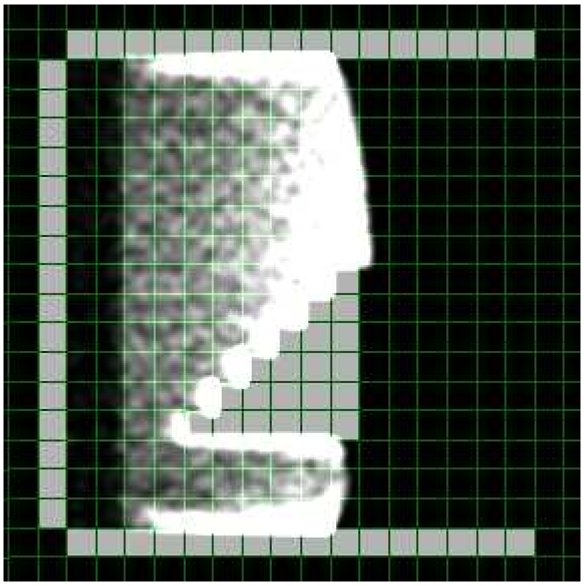

Figure 3.4: A flow moving along both a smooth and jagged surface (grid resolution 22×22 and particle size 0.2).

ranging from 1/8 to 1/3.

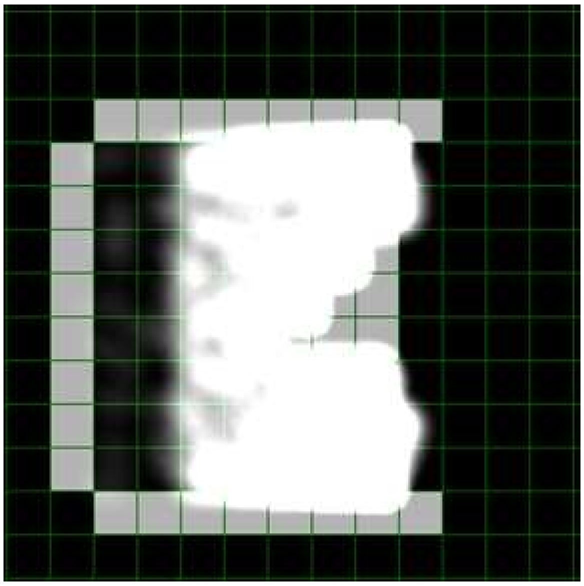

Figure 3.5: A flow moving along both a smooth and jagged surface (grid resolution 11×11 and particle size 0.4).

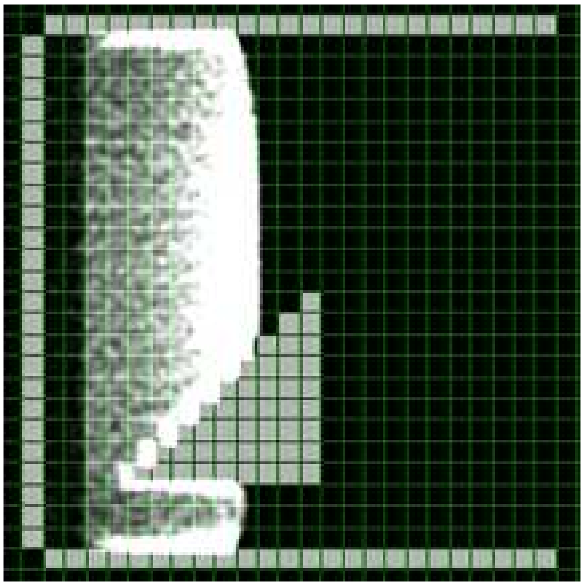

Figure 3.6: A flow moving along both a smooth and jagged surface (grid resolution 33×33 and particle size 0.4).

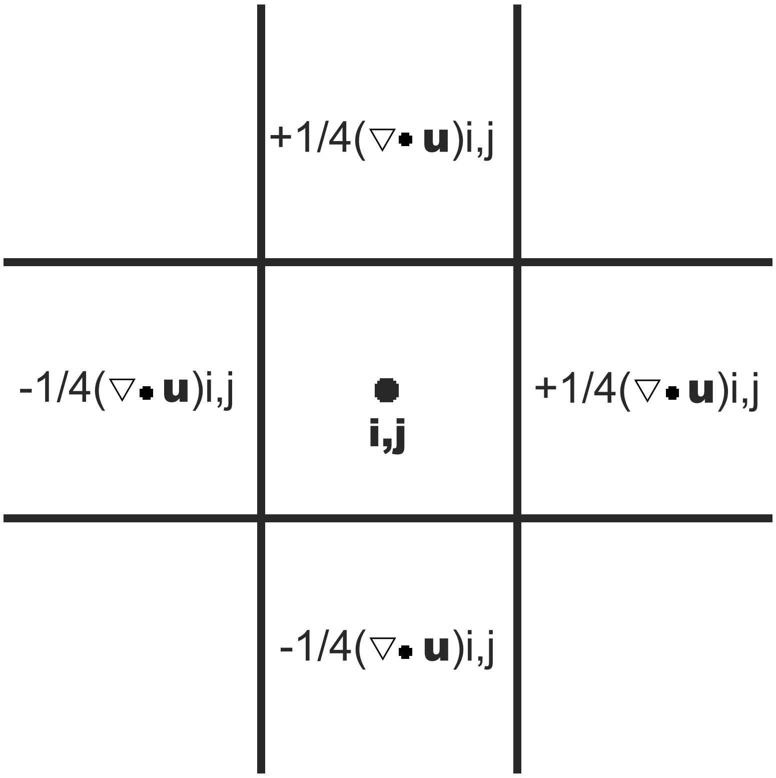

Figure 3.7: Schematic diagram of Buck's method shown here for a two-dimensional grid.

Chapter 4

Implementation and Results

The animations were modeled by approximating the solution to the Navier-Stokes equations for incompressible fluid flow as described in chapters 2 and 3. The computer code to perform these calculations executes as follows:

1. Compute the diffusion (2.25) and convection (2.26) components of the temperature distribution.

2. Compute the diffusion (2.7) and convection (2.14) components of the acceleration equation (2.1).

3. Add in computed thermal buoyancy (2.23).

4. Add in computed pressure gradient (2.18). For the method of Foster and Metaxas and the modified version discussed in chapter 3, computing the pressure gradient requires calculating a potential Ψ field, and then iterating over the grid using (2.18)

until convergence is achieved as determined by (2.19). For the method of Buck, the grid is also iterated over but the gradient is smoothed by subtracting 1/6 of the voxel's divergence from the lagging face and adding 1/6 of the divergence to the leading face (see figure 3.7).

5. The face velocities of each voxel are updated using (2.22), and the massless particles used to represent smoke are updated accordingly.

6. The code repeats from step 1 until the animation completes.

In order to illustrate the usefulness of the Navier-Stokes equations for incompressible fluid flow in physics-based modeling, several of the special effects first demonstrated by Foster and Metaxas in [7] have been reproduced in this chapter using the animation technique described in chapter 3. All of the results pictured in this chapter utilized the method of Buck [13] for mass conservation; however both the method of Foster and Metaxas and the modified method of Foster and Metaxas yield comparable results. The software for these demonstrations was written in the C language utilizing the OpenGL graphics library, and all results were generated on an AMD Athlon XP 2700+ 2.17 GHz PC with 512Mb of memory. Table 4.1 gives the specific parameters used to produce each animation. All examples had a constant ambient room temperature of 28.0 degrees Celsius, boundary temperatures on the grid of 28.0 degrees Celsius, a gravity setting of 0.98 m/s^2, and a particle size of 0.4 voxels.

4.1 Steam room animation

The steam room animation demonstrates the interaction between two jets of steam as they converge and mix into one gaseous cloud. The steam valves were created by aligning sets of voxels to form the approximate shape of a nozzle (see Figure 4.1), with appropriate temperature and velocity values assigned to each voxel so that they would act as solid objects. The jet of steam was produced by setting the voxels inside each nozzle to have a temperature of 80.0 degrees Celsius and a speed of 0.35 m/s, each in opposing directions. Massless particles are then released from those voxels. The result clearly shows the jets of steam from the nozzles mixing to form a single cloud of steam that slowly billows toward the ceiling (see Figures 4.2, 4.3, and 4.4). See Table 4.1 for details on resolution, time step, viscosity, diffusion, and thermal expansion settings.

Animation	Voxel resolution	Time step Δt (s)	Viscosity ν	Diffusion λ	Thermal expansion β
Steam Room	$15 \times 21 \times 21$	0.9	0.01	0.09	0.001
Smokestack	$21 \times 21 \times 21$	0.7	0.00001	0.3	0.01
FCS	$21 \times 21 \times 12$	0.6	0.0001	0.04	0.01

Table 4.1: Specific parameters used to produce each animation.

4.2 Smokestack animation

The smokestack animation shows smoke leaving a smokestack into a gusting wind. To add turbulence to the scene, voxels were set to mimic wind gusting by repetitively cycling through an array of values starting at 0.1 m/s, increasing to 0.2 m/s, 0.3 m/s, and then repeating at 0.1 m/s. The smokestack was created by placing sets of voxels together forming the shape of the stack (see Figure 4.5) with each voxel set

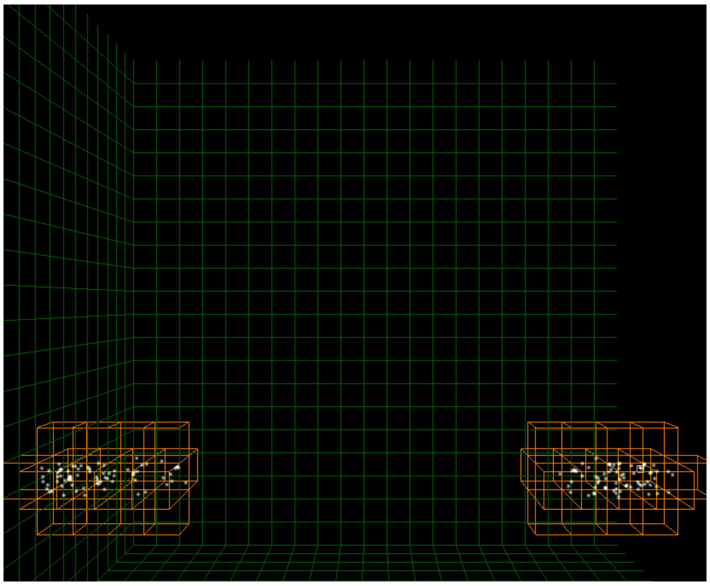

Figure 4.1: The orange framework depicts the voxels that comprise the steam nozzles.

to mimic the conditions of a real smokestack. The smoke was created by releasing massless particles from voxels inside the smokestack, each having a temperature of 46.0 degrees Celsius and an initial velocity of 0.8 m/s. When the rising motion of the smoke reaches the gusting wind, the forces combine to form the believable scene of flowing smoke pictured in Figures 4.6, 4.7, and 4.8. See Table 4.1 for details on resolution, time step, viscosity, diffusion, and thermal expansion settings.

4.3 Object Interaction animation

The FCS animation demonstrates the interaction of smoke with solid objects in the form of the block letters FCS (Faculty of Computer Science). The letters were created by defining voxels to form their desired shapes (see Figure 4.9) and then assigning temperature and velocity attributes consistent with those of solid objects at an ambient room temperature. The smoke is set to rise by assigning voxels below the letters with a temperature of 50.0 degrees Celsius, and both an upward vertical and leftward

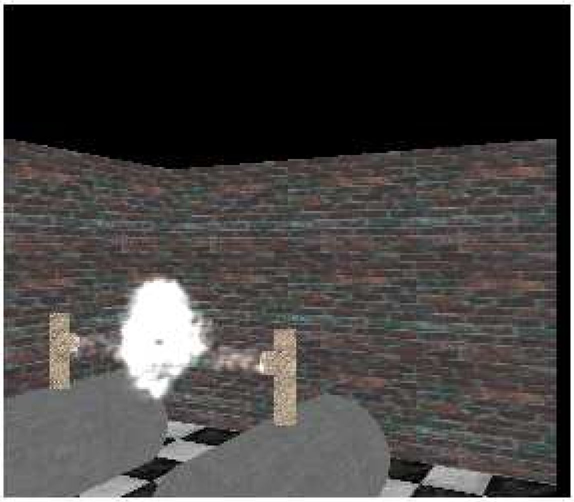

Figure 4.2: Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 150.

horizontal velocities of 0.1 m/s. This creates diagonal motion which makes for an interesting flow of smoke around the letters (see Figures 4.10, 4.11, and 4.12). Because the voxels governed by the solid letter objects dictate a velocity of zero, it is possible for a smoke particle to enter a solid object's voxels and become motionless. To counter this, smoke particles are kept in motion by identifying the particles that have drifted within a solid object's region, and then relocating them into the flow of smoke. A distance equal to 10% of voxel side was found by experimentation to be large enough to keep the smoke moving smoothly around the solid objects while still being small enough to ensure believability. See Table 4.1 for details on resolution, time step, viscosity, diffusion, and thermal expansion settings.

4.4 Discussion

This thesis has discussed two improvements to the mass conservation method of Foster and Metaxas (FM), a modified version of the Ψ field calculation in [7] (Mod. FM),

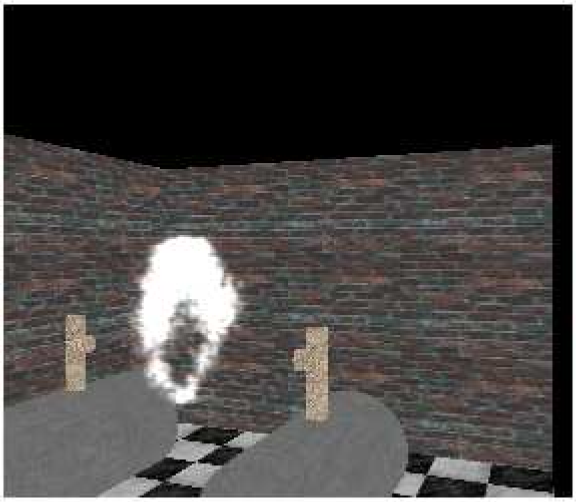

Figure 4.3: Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 325.

and a generalization of Buck's mass conservation [13] to three dimensions. In order to compare the improvements to the original method of Foster and Metaxas, profiles of execution time for the main functions of the animation software were done. The main functions are:

1. Diffusion and Convection for Temperature Distribution,
2. Diffusion and Convection for Acceleration Equation,
3. Mass Conservation, and
4. Rendering.

In addition, the total execution time of animation was also recorded. See Tables 4.2, 4.3, and 4.4 for the results of all three of the animations.

The functions for diffusion and convection for the temperature distribution and acceleration equation both showed no change in computation time. This is as expected because all the animations utilized the same functions to accomplish those computations.

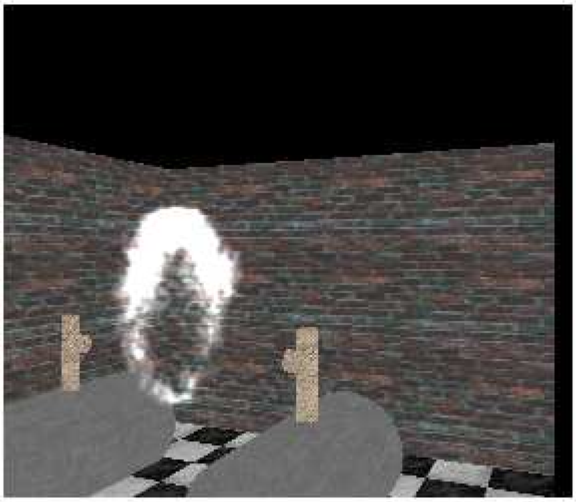

Figure 4.4: Steam jetting from two nozzles and converging to form a single cloud of steam at Frame 500.

Rendering is comprised of updating smoke particle positions and drawing them to the screen. All animation methods utilized the same rendering code, and although results varied slightly, the differences were negligible and likely associated with the computer hardware.

Function	FM	Mod. FM	Buck
Diffusion and Convection for Temperature Distribution	0.52	0.52	0.52
Diffusion and Convection for Acceleration Equation	1.95	1.95	1.95
Mass Conservation	4477.67	974.91	2.29
Rendering	2803.33	2804.16	2804.49
Total Execution Time	7284.79	3781.85	2810.53

Table 4.2: Profiling of the Steam Room animation as recorded in seconds.

Apart from the rendering process, which averaged just over 65% of the total computation time, the mass conservation method was by far the most expensive function

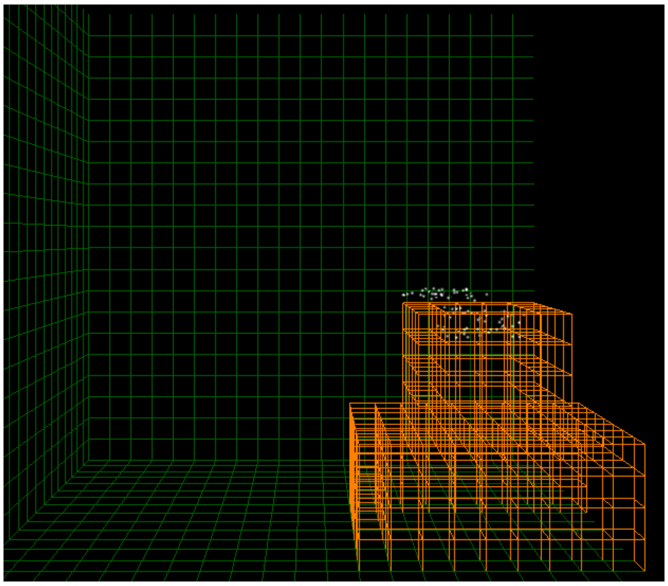

Figure 4.5: The orange framework depicts the voxels that comprise the smokestack.

Function	FM	Mod. FM	Buck
Diffusion and Convection for Temperature Distribution	0.16	0.16	0.16
Diffusion and Convection for Acceleration Equation	0.61	0.61	0.61
Mass Conservation	688.22	370.02	18.03
Rendering	877.71	879.43	912.93
Total Execution Time	1567.25	1250.76	932.27

Table 4.3: Profiling of the Smokestack animation as recorded in seconds.

averaging close to 35%. For all animations the profiler confirmed a decrease in total execution time based entirely on the modifications made to the mass conservation function.

Profiling of the steam room animation showed that the original method of Foster and Metaxas takes 4.6 times longer to converge than the modified method of Foster and Metaxas. It is nearly 2000 times slower than the generalization of Buck's mass conservation method [13].

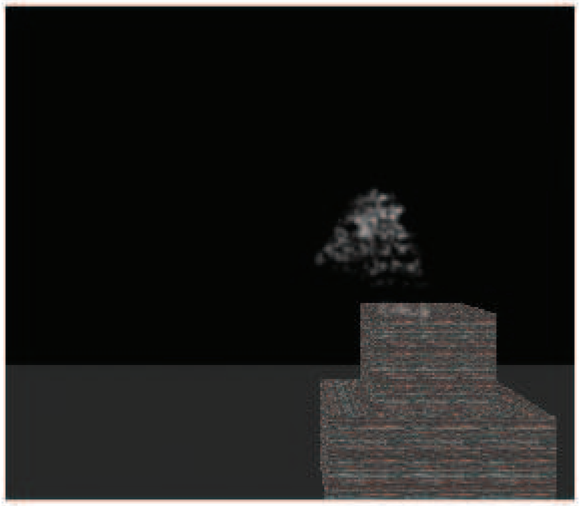

Figure 4.6: Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 60.

Profiling of the smokestack animation revealed that the original method of Foster and Metaxas took almost twice as long as the modified method of Foster and Metaxas and just over 38 times as long as generalized Buck method.

Finally, profiling of object interaction animation revealed that the original method of Foster and Metaxas took slightly more than 7 times as long as the modified method of Foster and Metaxas and just under 92 times as long as the generalized Buck method.

The modified method of Foster and Metaxas was on average 5.2 times faster than the original method of Foster and Metaxas. The generalized Buck method was clearly the fastest of all three methods, averaging 695.1 times faster than the original method of Foster and Metaxas and 152.9 times faster than the modified method of Foster and Metaxas.

Through experimentation 1/6 was also found to be the optimal coefficient for the

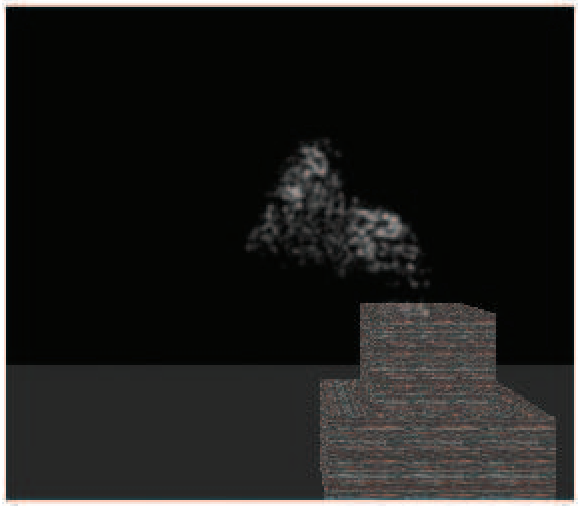

Figure 4.7: Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 80.

generalized method of Buck. See Table 4.5 for the results using other coefficients.

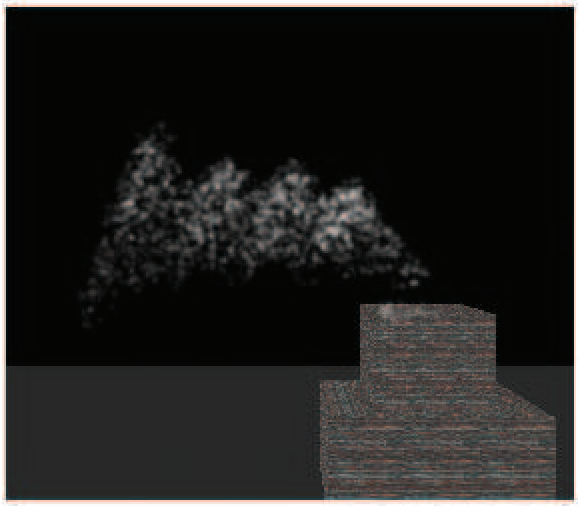

Figure 4.8: Smoke from a smokestack becomes turbulent as it interacts with a gusting wind at Frame 100.

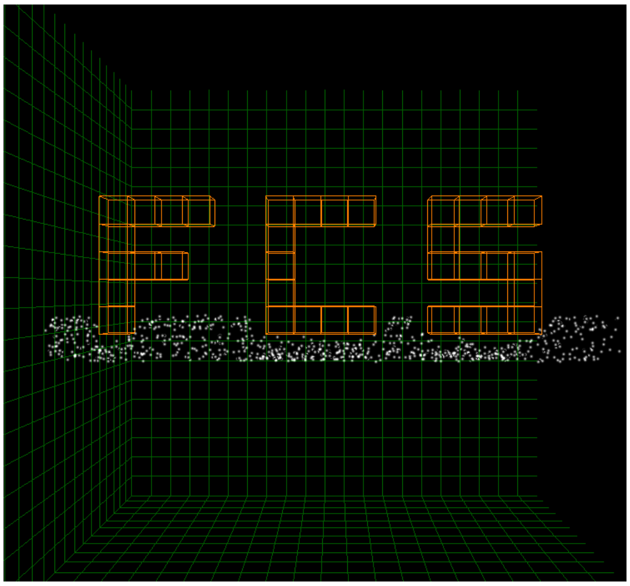

Figure 4.9: The orange framework depicts the voxels that comprise the letters FCS.

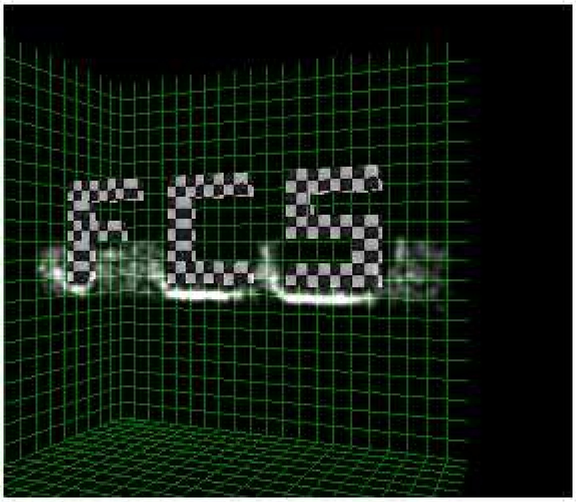

Figure 4.10: Smoke drifting around solid objects in the form of the letters FCS at Frame 20.

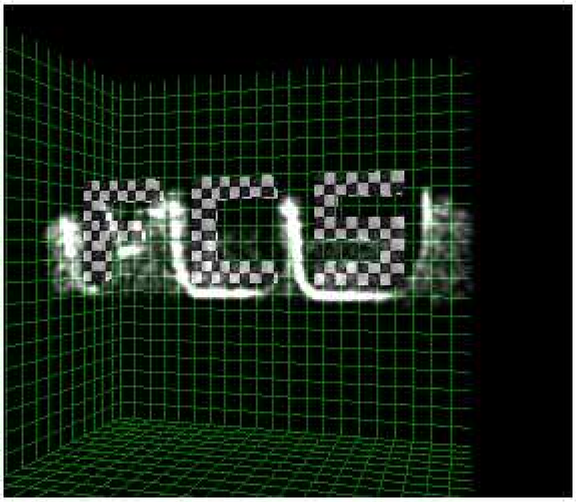

Figure 4.11: Smoke drifting around solid objects in the form of the letters FCS at Frame 50.

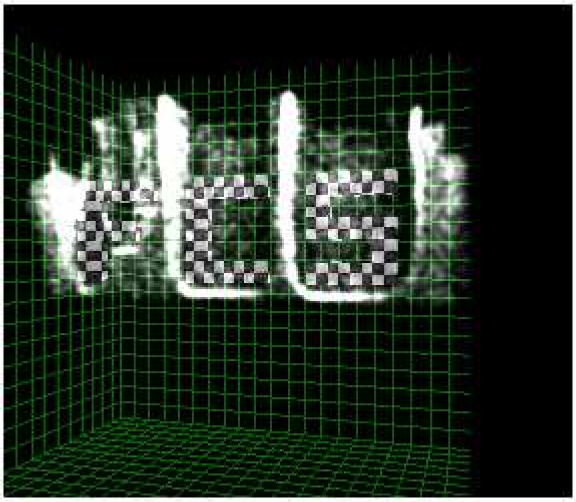

Figure 4.12: Smoke drifting around solid objects in the form of the letters FCS at Frame 150.

Function	FM	Mod. FM	Buck
Diffusion and Convection for Temperature Distribution	0.17	0.17	0.17
Diffusion and Convection for Acceleration Equation	0.63	0.63	0.63
Mass Conservation	823.94	112.01	8.97
Rendering	1003.33	1003.78	1002.98
Total Execution Time	1828.39	1116.91	1013.06

Table 4.4: Profiling the object interaction animation as recorded in seconds.

Mass convergence coefficient	Steam Room (iterations)	Smokestack (iterations)	FCS (iterations)
1/3	303,319	123,356	121,364
1/4	3,758	26,328	16,044
1/5	3,533	24,270	15,238
1/6	3,294	21,436	14,976
1/7	3,527	28,714	15,014
1/8	3,748	31,570	16,053
1/9	4,324	34,676	16,945

Table 4.5: Mass convergence coefficients for the Steam Room, Smokestack, and FCS animations.

Chapter 5

Conclusion

Physics-based modeling is prevalent in many fields of science and engineering as well as in the entertainment industry. However, entertainment applications such as special effects for movies or video games differ from science and engineering applications due to the lower requirement for accuracy. As long as the animation sequence is believable to an audience then the simulation is a success. Understanding that a believable simulation does not require the same level of accuracy as an engineering application allows for finding ways to decrease processing time by *approximating* the required physics.

Another important use of physics-based modeling is automation. As an animation project increases in complexity, a point is eventually reached where producing a believable simulation unguided by physics becomes impractical. This could result because a project is too complicated for an animation artist to handle, or because the project would take too long to complete by the unaided hand. Even for less complex simulations where accuracy is not important, mathematics and physics can still play an important role by automating the process, saving valuable human design time. In the case of modeling the motion of fluid-like phenomena such smoke or steam, the simulations are often based on a solution of the incompressible Navier-Stokes equations for incompressible fluid flow.

This thesis has taken an in-depth look at the approximation method developed by Foster and Metaxas for solving the Navier-Stokes equations for incompressible fluid flow with thermal buoyancy. As part of the examination, three special effects from Foster and Metaxas [7] were reproduced: a steam room, a smokestack, and an object with smoke moving around it (the block letters FCS). The attempt at reproducing their results was quite successful, and some encouraging results have been achieved and discussed in this thesis.

The steam room animation demonstrated how two high-velocity jets of steam interact to eventually form one large billowing cloud of vapour. The smokestack animation demonstrated how warm smoke directed by a smokestack interacts with a light variable wind. The block letters FCS animation demonstrated a smoke flow moving smoothly as it interacts with the solid FCS objects.

One drawback to the method of Foster and Metaxas was that their solution for mass divergence was slow to converge. This was addressed by adapting their potential field Ψ to start iterating using Ψ values computed from the previous iteration cycle instead of starting at zero. Although this did lower convergence time, instability of their method was still an issue. Again, this related to their method for computing conservation of mass. Foster and Metaxas later addressed this problem and released an addendum suggesting a relaxation method to be used in conjunction with their Ψ field; however even this did not always work due to convergence failure [14]. Eventually it was found through experimentation that a three-dimensional generalization of Buck's method [13] for mass conservation provided a stable substitute for their method. Instead of being based on a classic three-dimensional Poisson equation [7], Buck's method was to smooth the pressure gradient by subtracting a fraction of velocity from the lagging face and adding that amount to the leading face. Buck's method was originally developed for two dimensions, moving the divergence toward zero by subtracting 1/4 of the voxel's total pressure gradient from the lagging face and adding 1/4 of the gradient to the leading face. When generalizing Buck's method for three dimensions, 1/6 was used in place of 1/4 because it was found to be the optimal quantity (see Table 4.5 in chapter 4).

5.1 Future Work

There are still many problems inherent to the explicit finite difference method of Foster and Metaxas. For instance, their method is incapable of modeling fast-moving flows that go further than the side length of a single voxel per time step. The solution provided by Foster and Metaxas [7] is to simply decrease the time step, effectively making for a shorter particle movement. Although this is somewhat effective, it also adds considerably to computation time; however this is unavoidable in an explicit finite difference framework. A more suitable approach is to exchange the *explicit* method for an *implicit* method such as the semi-Lagrangian approach by Stam [8]. This approach cannot only model fast flows, but it is also more stable than explicit methods such as that of Foster and Metaxas.

Another shortcoming related to voxel size is that although a particle's position within a single voxel can be approximated through linear interpolation, rotational motion cannot be resolved on length scales equal to or less than the side length of a single voxel [7]. It has been stated in [7] that increasing the resolution of the entire grid will eventually produce visually acceptable results, and that an advanced solution utilizing an adaptive grid could give acceptable results without the overhead of increasing the resolution for the whole grid. Although this approach could be somewhat successful, rendering time would increase along with an increased demand on CPU and memory.

Another problem is that results will vary according to the orientation of particle flow through the grid. For example, a flow moving diagonally through the grid will tend to exhibit exaggerated turbulence. This is due to an increase in numerical diffusion perpendicular to the flow. It has been suggested [7, 14] that one should try to align animations parallel with the grid, but this may not always be possible. Perhaps experimenting with different shaped voxels such as octagons or hexagons would allow for less turbulence than the current cubic voxels. This could allow for more choice when trying to align with the grid, although at the cost of increased overhead.

Ultimately, the most important consideration is how to achieve desired believable effects with the least amount of computational overhead. This inevitably leads to furthering mathematical algorithms to improve overall efficiency. To this end, future work will almost surely abandon the explicit finite difference approximation method with its inherent deficiencies and instead explore more recent methods such as implicit semi-Lagrangian schemes, which are faster and more stable [8, 9, 10]. In fact, future work may abandon the Navier-Stokes equations altogether in favour of the incompressible Euler equations, which may arguably be more suitable and less expensive to solve than the Navier-Stokes equations [10].

Bibliography

[1] Paramount, *Genesis Demo from Star Trek II: The Wrath of Khan*, SIGGRAPH Video Review Number 11, June, 1982.

[2] Lucasfilm Ltd., *The Adventures of Andre and Wally B*, (film), August, 1984.

[3] Harlow, F.H. and Welch, J.E., *Numerical Calculation of Time-Dependent Viscous Incompressible Flow*, The Physics of Fluids, Volume 8, Number 12, (1965), 2182-2189.

[4] Reeves, W., *Particle Systems- A Technique for Modeling a Class of Fuzzy Objects*, ACM Transactions on Graphics, Volume 2, Number 3, (1983), 101-119.

[5] Reeves, W. and Blau, R., *Approximate and probabilistic algorithms for shading and rendering structured particle systems*, Computer Graphics, Volume 19, Number 3, (1985), 313-322.

[6] Stam, J. and Fiume, E., *Depiction of Fire and Other Gaseous Phenomena Using Diffusion Processes*, ACM SIGGRAPH, (1995), 129-136.

[7] Foster N. and Metaxas, D., *Modeling the motion of a hot, turbulent gas*, ACM SIGGRAPH, (1997), 191-199.

[8] Stam, J., *Stable fluids*, ACM SIGGRAPH, (1997), 1-8.

[9] Stam, J., *Interacting with smoke and fire in real time*, Communications of the ACM, Volume 54, Number 8, (2000), 86-94.

[10] Roland, F., Stam, J., and Jensen, H., *Visual Simulation of Smoke*, ACM SIGGRAPH (2001), 12-17.

[11] Foster, N. and Fedkiw, R., *Practical Animation of Liquids*, ACM SIGGRAPH, (2001), 15-22.

[12] Xiaoming, W., Li, W., and Kaufman, A., *Melting and Flowing of Viscous Volumes*, Computer Animation and Social Agents, (2003), 54-59.

[13] Buck, I., *Simulation of Turbulent Flow in Computer Graphics Applications and Optimizations*, http://graphics.stanford.edu/˜ianbuck/proj/Smoke/, (1998).

[14] Engell-Nielsen, T., *Modelling, Animation, and Visualization of Fire*, Thesis, Computer Science Department, University of Copenhagen, Denmark, April (1999).

[15] Foster, N., Errata for the *Modeling the Motion of a Hot, Turbulent Gas*, article by Nick Foster and Dimitri Metaxas. http://www.cis.upenn.edu/˜fostern, (2003).

[16] Foster, N. and Metaxas, D., *Realistic animation of liquids, Graphical Models and Image Processing*, Volume 58, Number 5, (1996), 471-483.

[17] Wei, X., Wei L., and Kaufman, A., *Melting and Flowing of Viscous Volumes*, Proceedings of the 16th International Conference on Computer Animation and Social Agents, IEEE Computer Society (2003), 54-59.

[18] Seaid, M., *Semi-Lagrangian Integration Schemes for Viscous Incompressible Flows*, Computational Methods in Applied Mathematics, Volume 2, Number 4, (2002), 392-409.