AN INVESTIGATION OF FOURIER DOMAIN FLUID SIMULATION

by

Soleil Lapierre

B.Sc., University of Calgary, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
August 2003

APPROVAL

Name:	Soleil Lapierre
Degree:	Master of Science
Title of thesis:	An Investigation of Fourier Domain Fluid Simulation
Examining Committee:	Dr. Ted Kirkpatrick Chair
	Dr. Torsten Möller, Assistant Professor, Computing Science Simon Fraser University Senior Supervisor
	Dr. Steven Ruuth, Assistant Professor, Applied Mathematics Simon Fraser University Supervisor
	Dr. Richard (Hao) Zhang, Examiner, Assistant Professor, Computing Science Simon Fraser University
	Dr. Oleg Veryovka, Scientist, Electronic Arts of Canada Supervisor
Date Approved:	AUGUST 13, 2003

An Investigation of Fourier Domain Fluid Simulation

Abstract

Motivated by the reduced rendering cost of the Fourier Volume Rendering method, we construct a Navier-Stokes fluid flow simulation that operates entirely in the frequency domain. We show results from a practical implementation and compare with Jos Stam's spatial domain and FFT-based simulations.

We break down the simulation pipeline into its major components and evaluate the cost of each component in the spatial domain and the frequency domain. We present an analytical as well as an experimental analysis, and we use our experimental results to identify the most efficient simulation pipelines for all grid sizes.

We conclude that frequency domain implementation of the Navier-Stokes flow equations for Euler modeling schemes is prohibitively expensive in terms of compute time. We also conclude that of the solutions we evaluated, Stam's semi-Lagrangian simulation pipelines are the best choices for real-time applications such as video games, where perfect physical accuracy is not required.

To the hydrofoil!

"It's more fun to compute."

— Kraftwerk, COMPUTER WORLD, *1981*

Acknowledgments

This thesis and my Master's degree would not have been possible without the support of others. Thanks to everyone who supported me emotionally, morally and financially during this effort, as well as those who assisted with my work.

- My fellow Nerd Hive members helped to keep me somewhat socialized during my graduate years.
- The SFU Photographers' Corner club provided a much-needed incentive to go out into the Big Room on a regular basis.
- The GrUVi lab crowd provided a positive work and research environment.
- Ken Chidlow brought fun to the lab and was a great sounding board for gripes.
- Torsten Möller, Reza Entezari, Steven Ruuth and Ramsay Dyer helped me with the math.
- Paul Stark and Steve Kilthau provided the Fourier Volume Rendering code for my software, Paul provided useful tips on how to make it work, and Reza Entezari extended it to implement lighting effects.
- Steve Kilthau and Reza Entezari provided me with important insight into the nature of grad school and research.

Special thanks to Torsten Möller for being the best supervisor a grad student could ask for.

I'd also like to thank musicians everywhere for doing what they do. Music is an important part of my life and I can't imagine life without it.

Last but not least, thanks to my parents, Eva-Anne and Maurice, for being supportive above and beyond the call of duty.

Financial assistance during my MSc program was provided by, in no particular order: Simon Fraser University, Torsten Möller, Liam Stitt, Sean Hawthorne, my parents, Tom Shermer, the B.C. New Media Innovation Centre (NewMIC), Electronic Arts of Canada, Radical Entertainment, the B.C. Advanced Systems Institute (ASI), the Alberta and Canada student loan programs and Citibank of Canada.

Contents

List of Figures

Chapter 1

Introduction

Fluid flow simulation is an interesting and challenging problem. Physically accurate prediction of the behavior of liquids and gases is valuable for applications such as weather forecasting, aerodynamic evaluation of vehicles, and special effects for movies and video games.

We observe that interactive, realistic flow simulations are virtually nonexistent in video games at the time of this writing. Fog, clouds, smoke, flames, dust and assorted liquids are common features of many different kinds of video games, and it is desirable to make them interact more realistically with moving objects in the game.

The current state of the art in interactive flow effects for video games is particle systems. They work well for limited applications such as localized flames and smoke plumes, but their utility is limited when a larger scope is needed, such as in weather or liquid simulation. A volumetric approach is more appropriate for these tasks.

There are two obvious barriers to acceptance of physical, volumetric flow simulations in the video game industry. One is the computational cost of such a simulation, and the other is the cost of rendering the output of the simulation.

While it is unlikely that the computational cost of a fully 3D simulation can be reduced below $O(n^3)$, we can take advantage of a fast volume rendering technique to reduce the total cost of simulation and rendering. Malzbender's Fourier Volume Rendering (FVR) technique [11] and subsequent work conducted in our lab [22] reduce the rendering cost to $O(n^2 \log n)$. Any 3D spatial-domain rendering technique requires $\Omega(n^3)$ time to render the same amount of data.

In order to capitalize on the FVR technique, which renders directly from the 3D Fourier domain to 2D screen space (see Figure 1.1), it is necessary to produce a flow simulation that operates in the frequency domain.

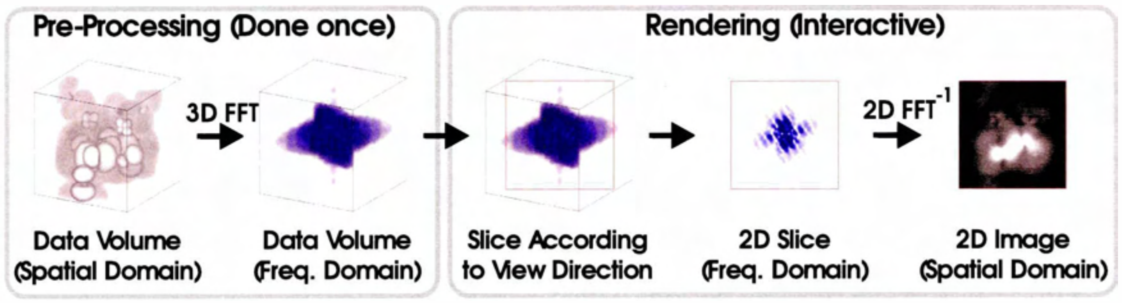
Pre-Processing (Done once)
Rendering (Interactive)
3D FFT
2D FFT^{-1}
Data Volume (Spatial Domain)
Data Volume (Freq. Domain)
Slice According to View Direction
2D Slice (Freq. Domain)
2D Image (Spatial Domain)

Figure 1.1: Fourier Volume Rendering Conceptual Model. Simulation data is transformed into the frequency domain (left side). Rendering is accomplished by extracting and transforming a 2D slice (right side). Figure courtesy Paul Stark and Steve Kilthau.

In this report we use the phrase "frequency domain" interchangeably with "Fourier domain" to mean the specific frequency representation produced by the Fourier transform.

Stam has shown [18] that it is advantageous to do some steps of a numerical flow simulation in the frequency domain. Stam's simulation is an approximation to the Navier-Stokes equations that describe natural fluid flow. It employs a semi-Lagrangian advection scheme in the spatial domain, then switches to the frequency domain to perform diffusion and mass conservation. Diffusion is simply a low-pass filter, and mass conservation is easily performed with a projection operation. The simulation data is then returned to the spatial domain for the next iteration. The asymptotic cost of Stam's simulation was $O(n^3 \log n)$ because it was necessary to Fourier transform all of the data twice per simulation step. We intended to eliminate the cost of the transformations by placing the entire simulation loop in the frequency domain.

We set out to transform the remaining spatial domain advection step of flow simulation into the frequency domain, and study the practical advantages and disadvantages of the result. This thesis report documents the results of the investigation. To the best of our knowledge, no attempt has been made to solve the equations of fluid motion fully within the frequency domain. The closest efforts we are aware of are that of Koster et al [9], who present an adaptive wavelet solver, and Stam [18], who does two of the simulation steps in the frequency domain.

If it is possible to produce an accurate (or sufficiently convincing) flow simulation entirely in the frequency domain at a reasonable computational cost, then we can leverage the lower rendering cost of FVR in order to make a less computationally expensive simulation. This could be applied to make volumetric flow simulation more practical for video games.

We also investigated frequency domain lighting effects for translucent, amorphous volumes such as fog and smoke. We found that some basic effects (linear shading and cylindrical light beams) are easy to compute, and we suspect there are others.

The remainder of this thesis is structured as follows: Chapter 2 introduces basic concepts of fluid dynamics and fluid simulation and ends with the derivation of the frequency domain expression for advection. Chapter 3 describes and structure of the simulation software we implemented to test our ideas and measure performance, and presents some of the visualizations we used to observe the results. Chapter 4 presents our performance measurements for the simulations we studied. Conclusions are presented in Chapter 5. Finally, Appendix A presents some lighting effects we combined with FVR, and Appendix B discusses boundary condition issues as preparation for future work.

1.1 Related Work

In 1993, Stam and Fiume [20] proposed a frequency synthesis method for adding small-scale detail to a coarse fluid simulation without significantly increasing the simulation size and cost. Their method was intended more for visual appeal than physical accuracy.

Later, Stam introduced an unconditionally stable, real-time fluid solver [18] and also showed how to perform fluid viscosity and mass conservation calculations in the frequency domain. More recently, Stam also published a stable real-time solver that works entirely in the spatial domain [17, 19] but we will show that it is slightly less efficient than his earlier work due to the choice of mass conservation solver.

Evangelinos and Karniadakis [3] discuss efficient parallel solution of the Navier-Stokes flow equations, and take it for granted that the nonlinear terms must be solved in the spatial domain. They studied the existing *Prism* solver, which, like Stam's work in [18], solves the linear terms in the frequency domain. Physically accurate fluid solvers have also been the focus of research by Fedkiw and collaborators [4, 13].

Witting [28] describes applications of fluid solvers to special effects in movies. He shows examples of fog and smoke effects, swirled soup and distortion of images all done with fluid simulations for the film *The Prince of Egypt.* Treuille et al [26] demonstrate a way of key-framing a flow simulator to give artists increased control over the results.

Fourier domain volume rendering (FVR), introduced by Malzbender [11], and extended by Totsuka and Levoy [25], is a direct volume rendering algorithm that is significantly different from all commonly used ones (for an overview of volume rendering algorithms see [12]). FVR is based on the Fourier projection-slice theorem [1], the relevant result of which is that any 2D slice through the origin of a frequency domain volume will contain the spectrum of an X-ray projection of the equivalent spatial domain volume. The algorithmic complexity for rendering an n^3 volumetric dataset is $O(n^2 \log n)$, after incurring a $O(n^3 \log n)$ pre-processing step if the data is given in the spatial domain. All other rendering algorithms are required to traverse the entire volume and hence have an algorithmic complexity of $\Omega(n^3)$. By suitably pre-multiplying the frequency data, one can also achieve depth cues and effects similar to diffuse lighting.

Entezari et al [2] have improved the approximate linear lighting model of Totsuka and Levoy [25] using spherical harmonics. The cost, however, is in storing nine additional frequency volumes.

Chapter 2

Fluid Flow Theory

2.1 Fluids in Nature

This chapter introduces basic fluid flow concepts and equations and leads up to the derivation of the frequency domain flow equations, which are the focus of our investigation. A short discussion of simulation stability issues concludes the chapter.

A fluid is defined as a substance that undergoes continuous deformation when subjected to a shear stress [8]. Types of fluid flows found in nature include gaseous, liquid and plastic flows.

Examples of gaseous flows are the planetary atmosphere and disturbances within it, such as are produced by propellers, jets, explosions, heat sources, storms and moving objects. Examples of liquid flows are water flowing down a stream bed, a drink being stirred, the flow of water around a boat and its propellers, the pooling of rain on irregular surfaces, and the mixing of molten metals to produce alloys. Examples of plastic flows are the deformations of semi-solid substances such as ice in glaciers, modeling clay and the subsidence or slumping of soft wax or earth.

Fluids are classified as being either Newtonian or non-Newtonian [8] according to whether or not they conform to Newton's law of viscosity, which states that the shear stress in a fluid is linearly proportional to the strain rate. Another way of saying this is that a Newtonian fluid reacts the same way to being deformed at different speeds.

Plastic flows are non-Newtonian because they do not start to flow until a certain minimum force is applied. Slurries like ketchup are non-Newtonian because their composition makes viscosity a nonlinear function of the force applied (it resists up to a point and then suddenly starts to flow easily).

This thesis is an investigation of the question of whether or not doing fluid simulation in the frequency domain is practical. We make several assumptions in order to have a simple starting point; simulating fluids in general is left for future research. The first simplification is that we will concentrate on modeling Newtonian fluids.

The study of flows, usually referred to as the discipline of fluid dynamics, is a huge research area with many practical applications. Fluids are essential for our life processes and play important roles in our everyday lives. Their prevalence in our environment and their profound, sometimes devastating effects make them obvious candidates for study, modeling and simulation.

There is a long history of literature on fluid modeling, and many mathematical descriptions have been developed for special cases and simplified conditions. The first mathematical formulation that fully describes the behavior of all Newtonian fluid flows is that discovered independently in the early 19th century by L. Navier and Sir G. G. Stokes. This Navier-Stokes formulation is described in the subsequent sections.

We first describe some of the properties and metrics of flows and flowing substances:

- *Viscosity* is the tendency of the substance to resist deformation. More formally, it is the coefficient that relates shear stress and rate of deformation. A thick liquid like glue has a very high viscosity. Gases have a very low viscosity. Liquids like water and oil fall somewhere in between. In a Newtonian fluid, viscosity is constant throughout a homogeneous volume.

  Viscosity is determined by the molecular properties of the material, and can be affected by temperature and pressure. Viscosity is not related to the density of the substance; motor oil is more viscous than water but is less dense, while honey is also more viscous and is more dense than water.

- *Compressibility* describes the ability of a substance to respond to pressure by becoming more dense. Ignoring extreme gravitational effects that can collapse the space between atoms, only gases are compressible. Some semi-solids like soil and rubber appear compressible, but that is because they include pockets of gas that are compressed or squeezed out when pressure is applied.

  As another simplification, we concentrate on incompressible flows. Flows that do not exhibit density changes or compression are said to be *mass-conserving*.

- *Turbulence* is something that arises in low-viscosity flows in response to obstructions or colliding flows. Turbulence is a form of instability that can sometimes cause undesirable side effects. A large scale example is tornados arising in the interface between differing air masses.

A small scale example is air bubbles interfering with the flow of water through pipes. Turbulence can either increase or decrease drag on objects within the flow, depending upon how it interacts with the object and the surrounding medium. This is why vehicle hulls are designed to shape turbulence in the air or water they pass through, and why golf balls have dimples.

Turbulence is very important for visual realism of flows, and a simulation without it would not enjoy wide application in graphics. Therefore we retain the ability to model turbulent flows.

- *Boundary conditions* are a special topic of study. Boundary conditions arise wherever a volume of flowing material meets a volume of some other material. An obvious example of a boundary is the surface of contact between a liquid and its container, or between a stream and the stream-bed. Less obvious cases of boundary conditions are the regions where liquids and gases meet and where immiscible (non-mixing) fluids like oil and water interact.

  An interesting observation about boundaries between fluids and solids is that on the small scale, the liquid velocity at the boundary is equal to the velocity of the solid. There is a velocity gradient near the surface that is related to the fluid viscosity and can be used to measure the viscosity of an unknown fluid. This gradient causes shear stress on the fluid near boundaries, which can give rise to drag and turbulence.

  Because our goal is a realistic and interactive flow simulation, we retain the capability of interacting with solid objects. The topic of boundary conditions is discussed in more detail in Appendix B on page 49.

2.2 Continuous Formulation

Mathematical descriptions of flows appear in three main categories: Lagrangian, Eulerian and semi-Lagrangian.

The Lagrangian representation tracks individual packets of fluid and describes what happens to them over time. In other words, the velocity of a packet of fluid is a function of time and the packet's identity. This is the representation that is used in a particle system.

The Eulerian representation describes what happens at a particular point in space and time. Here fluid velocity is a function of time and spatial location, and we do not track the motion of individual volumes of fluid.

The Eulerian representation is more convenient for volume rendering because it provides an implicit ordering to the data and doesn't require additional storage for the sample coordinates. It is also useful for immersive flows such as video game weather simulations, where we are usually more interested in the behavior of the fluid at given locations in space than in knowing what happens to a particular volume of fluid. It is also more efficient for large-scale simulations because it is not necessary to calculate new coordinates for the sample points.

The semi-Lagrangian representation is a hybrid that allows the same convenient grid representation as the Eulerian approach and updates the grid using the Lagrangian particle-tracing idea. This is the method Stam used in [18] and is also known as the backward characteristic or CIR (Courant, Isaacson and Rees) scheme [24]. It can be thought of as a Lagrangian scheme wherein at each time step, we choose to model the set of particles that will arrive at convenient Cartesian grid points at the end of the interval [21].

Since our goal was to create a computationally convenient simulation for graphics applications, we focus on Eulerian and semi-Lagrangian approaches. We use the Navier-Stokes flow equations as a starting point, since they are the most accurate for common fluids.

In this thesis we use the following notation conventions:

- Spatial domain functions are represented with lowercase Roman letters, and their frequency domain counterparts with the same letters in uppercase. Examples: u, U.
- Parameters and constants are represented by lowercase Greek letters. Examples: ρ, μ.
- Operators used in our notation include the dot product, denoted by $\cdot$, the gradient operator ∇, the partial derivative ∂, the forward Fourier transform $\mathcal{F}$ and the reverse Fourier transform $\mathcal{F}^{-1}$.
- Coordinates are represented by x, y and z in the spatial domain, where they refer to spatial position. In the frequency domain, where coordinates refer to relative frequency in each of the spatial directions, they are labeled ω_x, ω_y and ω_z. In vector notation, $\vec{x} = [x, y, z]$ and $\vec{\omega} = [\omega_x, \omega_y, \omega_z]$. Time is always represented by t.
- Velocities are represented by u, v and w which refer to the velocity vector components in the x, y and z directions respectively. The vector $\vec{u} = [u, v, w]$ and in the frequency domain, $\vec{U} = [U, V, W]$. Note that velocity is considered a function of location, so the notation given here is a short form for $\vec{u}(\vec{x}) = [u(\vec{x}), v(\vec{x}), w(\vec{x})]$ and $\vec{U}(\vec{\omega}) = [U(\vec{\omega}), V(\vec{\omega}), W(\vec{\omega})] = [\mathcal{F}(u(\vec{x})), \mathcal{F}(v(\vec{x})), \mathcal{F}(w(\vec{x}))]$.

The Navier-Stokes flow equations are commonly written as shown below. Here p is a pressure field, $[g_x, g_y, g_z]$ is the strength and direction of gravity, μ is the fluid viscosity and ρ is density.

$$\rho\left[\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}\right]=-\frac{\partial p}{\partial x}+\rho g_x+\mu\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right], \tag{2.1}$$

$$\rho\left[\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}\right]=-\frac{\partial p}{\partial y}+\rho g_y+\mu\left[\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right], \tag{2.2}$$

$$\rho\left[\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}\right]=-\frac{\partial p}{\partial z}+\rho g_z+\mu\left[\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right] \tag{2.3}$$

We make the simplifying assumptions that our fluid is incompressible, and that $\partial p = 0$ everywhere. We can then drop the terms involving p. It is a common mistake to say that incompressibility alone allows the removal of the pressure factor. An incompressible fluid like water can still be under pressure, and will flow toward regions of lower pressure. We need the second assumption of uniform pressure in order to drop the pressure terms.

We will further assume that our fluid is of uniform density $\rho = 1$, which allows us to ignore ρ and drop the gravity term (gravity will not cause any motion unless some volumes are heavier than others). This leaves:

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}=\mu\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right], \tag{2.4}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}=\mu\left[\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right], \tag{2.5}$$

$$\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}=\mu\left[\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right] \tag{2.6}$$

For compactness, we rewrite these equations using vector notation and the gradient operator. Here the gradient is applied to scalar fields (the flow vector components) and produces a vector result.

$$\frac{\partial u}{\partial t}+\vec{u}\cdot\nabla u=\mu\nabla^2 u, \tag{2.7}$$

$$\frac{\partial v}{\partial t}+\vec{u}\cdot\nabla v=\mu\nabla^2 v, \tag{2.8}$$

$$\frac{\partial w}{\partial t}+\vec{u}\cdot\nabla w=\mu\nabla^2 w \tag{2.9}$$

We define the gradient operator to be the column vector of partial derivative operators:

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} \tag{2.10}$$

The gradient of a scalar field is then the vector:

$$\nabla u = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \end{bmatrix} \tag{2.11}$$

The gradient of a vector field is a tensor formed by the matrix product of the gradient operator and the vector:

$$\nabla \vec{u} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} & \frac{\partial w}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} & \frac{\partial w}{\partial y} \\ \frac{\partial u}{\partial z} & \frac{\partial v}{\partial z} & \frac{\partial w}{\partial z} \end{bmatrix} \tag{2.12}$$

We use the shorthand $\nabla^2 \vec{u}$ to represent the diffusion terms as the following vector, written in column form for compactness:

$$\begin{aligned} \nabla^2 \vec{u}(\vec{x}) &= \nabla^T (\nabla \vec{u}) \\ &= \begin{bmatrix} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \\ \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \\ \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \end{bmatrix}^T \end{aligned} \tag{2.13}$$

These definitions allows us to rewrite the simplified Navier-Stokes equations as a single matrix equation:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}(\nabla \vec{u}) = \mu \nabla^2 \vec{u} \tag{2.14}$$

Equation 2.14 is the simplified Navier-Stokes formulation that we use as the basis for our discrete implementation and Fourier transformation in the next section.

2.3 Discrete Simulation

To do fluid simulation on a computer we can use the physical equations given in the previous section, but with the knowledge that sampling and accuracy issues come into play since we cannot do the simulation on an infinitely fine grid with infinitely small time steps.

For interactive applications, it is necessary to be able to incrementally update our simulation state over small time steps. Thus we are interested in solving for the term $\frac{\partial \vec{u}}{\partial t}$. Rearranging Equation 2.14 to solve for the temporal difference term, we get:

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u}(\nabla \vec{u}) + \mu \nabla^2 \vec{u} \tag{2.15}$$

The first term on the right hand side of Equation 2.15 is referred to as the *advection* term because its influence on the result is to push material around (advect it) in response to the flow velocity. The second term on the right hand side is the *diffusion* term and controls the smoothing out of small features as a result of viscosity and dispersion.

The Navier-Stokes flow equations by themselves are not enough to produce a realistic simulation. It is also necessary to enforce some physical laws including at a minimum the law of conservation of mass, which says that matter can neither be created nor destroyed.

$$\frac{\partial \rho}{\partial t} + \overline{\nabla}(\rho \vec{u}) = 0 \tag{2.16}$$

$$\overline{\nabla} \vec{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{2.17}$$

In the general case, mass conservation can be achieved by enforcing the condition given in Equation 2.16. In an incompressible flow, mass conservation simplifies to the condition of Equation 2.17. Here $\overline{\nabla}$ denotes divergence, defined as shown in Equation 2.17.

In [18], Stam observed that the mass conservation operation can be efficiently performed by a projection in the frequency domain, and that the diffusion operation can easily be performed at the same time by applying a simple low-pass filter. An existing mathematical result called the Hodge decomposition (cited by Stam in [19]) states that any vector field is the sum of a mass-conserving vector field and a gradient field. The advection operation produces a non-mass-conserving field, and from that we can extract the desired mass-conserving result by computing and subtracting the gradient field. In [17] Stam derives this as a projection operator and shows the conversion to the frequency domain. The explanation is enhanced somewhat in [19] although only for the spatial domain.

We compared the spatial domain and frequency domain mass conservation operations presented by Stam. Our experimental results (see Chapter 4) show that the frequency domain version is roughly an order of magnitude less expensive than the spatial domain version, but not asymptotically less expensive. In fact, if a Fourier transform is required in order to perform this operation in the frequency domain, it becomes asymptotically more expensive.

The main reason the frequency domain version is faster is that both the projection and the diffusion can be performed in a single pass through the data. In the spatial domain, the two operations use a Gauss-Seidel relaxation solver, which requires multiple passes in order to achieve convergence. Stam used a constant twenty iterations of the Gauss-Seidel solver based on the claim of Foster and Metaxas [5] that eight to twenty iterations is sufficiently accurate for graphics applications of fluid solvers. The Gauss-Seidel solver is normally considered to require a number of iterations dependent on the data size. If the number of iterations were not kept constant, the spatial domain mass conservation operation would likely be asymptotically more expensive than switching to the frequency domain.

Despite the higher asymptotic cost of the Fourier transform, we will show in Chapter 4 that it is still better to do diffusion and mass conservation in the frequency domain than to use the spatial domain solution mentioned above. However, we did not know this fact beforehand and so eliminating the Fourier transform cost was part of our motivation; doing all simulation steps in the frequency domain eliminates the need for two Fourier transforms per simulation step.

In [18], Stam applied the semi-Lagrangian advection technique to make his simulation unconditionally stable without significantly increasing the update cost. This replaces the highly localized advection term $\vec{u}(\nabla\vec{u})$ from Equation 2.15 with a new one. Instead of using the local gradient to indirectly determine how to update the local sample, a semi-Lagrangian scheme attempts to determine where in space the new value of the local sample will come from. It does this by linearly approximating a streamline backward in time from the current sample to some other location in space. The value at that location is interpolated and taken to be the new value of the sample being updated.

This approach has the advantage of making the numerical simulation unconditionally stable, and it is easy to see why: The new sample values are calculated by linear interpolation from the previous values. Linear interpolation never produces values outside the range of the input, which means that the energy in the simulation can never be increased by this operation. Higher-order interpolation can be used to increase accuracy, but if it does not have this property it can introduce instability.

The semi-Lagrangian scheme gives us the updated simulation state directly rather than in the form of an incremental change. We can write it as a follows:

$$\vec{u}(\vec{x})_{t+\partial t} = \vec{u}(\vec{x} - \partial t \vec{u}(\vec{x})) + \mu \nabla^2 \vec{u} \tag{2.18}$$

This formulation is no longer physically accurate, although it approaches the Navier-Stokes solution in the limit as the grid spacing and time step approach zero. However, its unconditional stability makes it very attractive for graphics and video game applications, where only a convincing approximation of physical accuracy is required.

The disadvantages of the semi-Lagrangian approach are that it does not propagate shockwaves at the right speed in a compressible fluid, and that its reliance on interpolation increases numerical dissipation. The high numerical dissipation causes vortices and fine features of the flow to damp down too quickly, although for graphics applications it is seldom a problem. If desired, techniques such as frequency synthesis [20] and vorticity confinement [23] can be used to add fine detail back in and counteract the damping effect. There are also non-interpolating schemes such as that by Ritchie [16] that are simultaneously more accurate than semi-Lagrangian interpolation and more stable than simple Eulerian methods, but are slightly more complex to compute.

Stam used the semi-Lagrangian advection scheme in the spatial domain coupled with the frequency domain diffusion and mass conservation operations in [18]. This was a logical starting point for our work because of its stability and because it was already partly in the frequency domain.

We found no direct transformation of the semi-Lagrangian advection method into the frequency domain. We took the first two terms of its Taylor series expansion as an approximation. The result can be transformed, and we later discovered that it was equivalent to the simplified Navier-Stokes equations. It therefore has greater physical accuracy than the semi-Lagrangian formulation but lost the desirable stability property.

We implemented a simplified Navier-Stokes simulation and found that although some inconvenient limitations are required to keep it stable (discussed in section 2.3.2), the appearance of the output is as appealing as that from Stam's simulations. We then transformed this spatial domain Navier-Stokes solver to work in the frequency domain.

As an aside, note that Equation 2.15 only describes self-advection of the flow velocity field by itself. It does not cover transportation of other quantities like fog vapor density, pressure, temperature and so on. However, the equation does generalize easily for such applications. Equation 2.19 shows an example for advecting an arbitrary scalar field f using the flow velocity field $\vec{u}$.

$$\frac{\partial f}{\partial t} = -\vec{u} \cdot \nabla f \tag{2.19}$$

The diffusion term can be omitted as shown here, or modified in accordance with the properties of the quantity represented by f. The enforcement of conservation properties is also left to the discretion of the user. In our implementation we omitted the diffusion and mass conservation operations for the fog density field. We even took steps to reduce the diffusion inherent in the simulation in order to retain maximum visibility of fog features. The semi-Lagrangian scheme can also be applied to arbitrary fields in this way. In general the Eulerian strategy works on any data type for which a gradient can be defined, and the semi-Lagrangian strategy works on any data type that can be interpolated.

2.3.1 Frequency Domain

In order to implement a frequency-domain fluid simulation pipeline, the most important task was to transform the Navier-Stokes advection term to the frequency domain.

Frequency-domain expressions of the viscosity (diffusion) term of the Navier-Stokes equation and of the mass conservation operation were previously provided by Stam in [18]. Ignoring those, the remaining advection term to be transformed into frequency space is shown in Equation 2.20.

$$\frac{\partial \vec{u}(\vec{x})}{\partial t} = -\vec{u}(\vec{x})(\nabla \vec{u}(\vec{x})) \tag{2.20}$$

Multiplying the vector field $\vec{u}$ by its gradient matrix gives the vector result shown in Equation 2.21. We write the vector in column form for compactness.

$$\vec{u}(\nabla \vec{u}) = \begin{bmatrix} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} \\ u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} \end{bmatrix}^T \tag{2.21}$$

We now establish the analytical Fourier transform of Equation 2.20 by transforming the terms of Equation 2.21. The relation between the Fourier transform of a function and its derivative can be shown to be:

$$\mathcal{F}\{\frac{\partial u(\vec{x})}{\partial x}\} = -i\omega_x U(\vec{\omega}) \tag{2.22}$$

where $U(\vec{\omega})$ denotes the Fourier transform of $u(\vec{x})$. Similar relations hold for the partial derivatives along the y and z axes and for the v and w components. Using this relation, we can derive the Fourier transform of the advection term $\vec{u}(\vec{x})(\nabla\vec{u}(\vec{x}))$:

$$\mathcal{F}(\vec{u}(\vec{x})(\nabla\vec{u}(\vec{x}))) = -i\begin{bmatrix} U\otimes\omega_x U + V\otimes\omega_y U + W\otimes\omega_z U \\ U\otimes\omega_x V + V\otimes\omega_y V + W\otimes\omega_z V \\ U\otimes\omega_x W + V\otimes\omega_y W + W\otimes\omega_z W \end{bmatrix} \tag{2.23}$$

$$= -i[\vec{U}(\vec{\omega})\otimes(\vec{\omega}^T\vec{U}(\vec{\omega}))] \tag{2.24}$$

Therefore,

$$\frac{\partial\vec{U}}{\partial t} = -i[\vec{U}(\vec{\omega})\otimes(\vec{\omega}^T\vec{U}(\vec{\omega}))] \tag{2.25}$$

Here $\otimes$ denotes convolution. Note that as with the spatial domain advection term, the left side of the convolution is a vector (a 1×3 matrix in a 3D simulation), and the right side is a 3×3 matrix. The convolution is performed using matrix multiplication to produce a 1×3 result vector, which is then simply added to the simulation state to produce the new state. Computing the result requires nine convolutions between the Fourier transformed components of the flow vectors and their gradients. If additional quantities such as fog density must be advected, three more convolutions are needed per scalar field. In our experiments in Chapter 4 we used a 3D flow simulation with a single density field, for a total of twelve convolutions.

The expression of the advection term in Equation 2.25 has the property that it can be combined with Stam's frequency-domain viscosity and mass conservation terms to achieve a simulation without Fourier transforms in its main loop. It can also be coupled to a Fourier Volume Renderer to achieve $O(n^2\log n)$ rendering time.

The price of achieving these two goals was high; the convolutions raise the simulation update cost from $O(n^3)$ to $O(n^6)$. It may be possible to perform the vector convolution in a manner optimal in terms of cache hits but this will not overcome the disadvantage of the high asymptotic cost. Further discussion of the convolution cost will follow in section 4.2

2.3.2 Stability

Stability remains an unsolved problem in our frequency domain simulation. Semi-Lagrangian schemes are stable unless higher-order interpolation is used, and stability conditions are known for spatial domain Euler schemes.

However, our simulation is a Fourier transformed Euler scheme and the known stability conditions require knowledge of the spatial domain velocities. The stability condition commonly given for Eulerian flow simulations is an inequality known as the Courant-Friedrichs-Lewy (CFL) condition [14]. The CFL condition for our spatial domain advection operation is shown in Equation 2.26. Here Δx, Δy and Δz refer to the sample spacing along each of the simulation grid axes.

$$\partial t < \frac{\min(\Delta x, \Delta y, \Delta z)}{\max(|u| + |v| + |w|)} \tag{2.26}$$

The interpretation of the CFL condition is that the time step must be limited so that we never try to obtain information from more than one grid cell away. The Eulerian advection operator is based on the velocity at the sample point and the gradient, both of which are spatially local. With this information it is not possible to accurately infer the contents of cells that are not immediate neighbors; we must wait for distant information to be propagated to us over time. It is not clear that there is an analytical Fourier transform of this CFL condition.

The straightforward simulation stepping method is to compute the change over some time interval and then add that difference to the previous state to get the new state. This is called Euler stepping. We also tried employing fourth-order Runge-Kutta stepping in order to increase stability. Runge-Kutta methods combine multiple estimates of the derivative to get a more accurate prediction of the system's state at the next time step. The properties of Runge-Kutta methods have been thoroughly analyzed in the simulation community and are beyond the scope of this report.

In our implementation stability was improved by using Runge-Kutta stepping. The main strength of Runge-Kutta methods is that they can increase accuracy and can thus allow the use of larger time steps without loss of stability. However, use of Runge-Kutta stepping does not guarantee stability and as Witting [28] observed, fourth-order Runge-Kutta stepping requires four times as many function evaluations as simple Euler stepping, but only allows time steps to be up to 2.82 times larger than Euler stepping with the same stability results. This is not beneficial since the gain is less than the expense, and is therefore not a generally suitable solution for real-time applications such as video games. The requirement for additional evaluations is especially bad given the high cost of the frequency domain simulation.

A further hindrance to achieving stability of the frequency domain simulation is that the previously mentioned CFL condition is necessary but not sufficient for stability. In order to obtain stability, we also had to use an *upwinding* technique when calculating the gradient and divergence. Central differencing, forward differencing and multiplication by wave-numbers (the Fourier expression of the gradient operator) were all insufficient to obtain stability in our experiments.

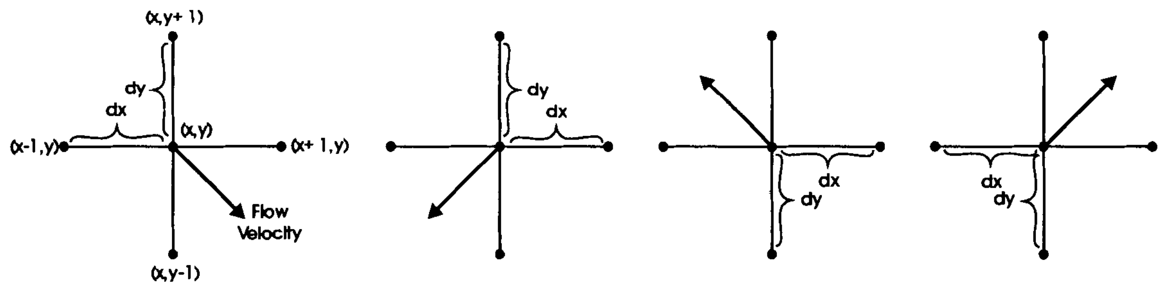

Figure 2.1: Upwind Gradient Calculation Cases for 2D. The diagonal line in each case indicates the flow direction. Gradients are calculated by differencing on the upwind side, along the grid edges marked dx and dy.

The idea behind upwinding is that information propagates with the flow direction, and therefore the needed data must be taken from upwind. The specific application to our Navier-Stokes simulation is that we use differences in the upwind direction from each sample point to approximate the gradient at that sample point.

Figure 2.1 shows the four cases for upwind differencing in two dimensions. The five dots in each case represent the sample point being considered and its four immediate neighbors. The diagonal arrow in each case represents the local flow direction. In each case we approximate the gradient using the central sample and the two samples bordering the quadrant *opposite* the flow direction - the upwind side of the central sample.

For each axis, a conditional test is made on the sign of the corresponding velocity component (u for the x axis, v for the y axis) to determine which quadrant the flow vector falls in. The difference for the gradient calculation is then taken from the quadrant opposite the wind direction. In the figure, dx and dy refer to the grid edges along which the difference is taken for each case. This is very quick to compute and only requires two conditional tests in 2D (three in 3D). Unfortunately upwinding requires spatially local information about the flow velocities, which is not available in the frequency domain.

$$\nabla\vec{u} = \left[\frac{\partial u}{\partial x}, \frac{\partial v}{\partial y}, \frac{\partial w}{\partial z}\right] \text{ where} \tag{2.27}$$

$$\frac{\partial u}{\partial x} = sign(u)\frac{[u(x,y,z) - u(x - sign(u)\partial x, y, z)]}{\partial x} \tag{2.28}$$

$$\frac{\partial v}{\partial y} = sign(v)\frac{[v(x,y,z) - v(x, y - sign(v)\partial y, z)]}{\partial y} \tag{2.29}$$

$$\frac{\partial w}{\partial z} = sign(w)\frac{[w(x,y,z) - w(x, y, z - sign(w)\partial z)]}{\partial z} \tag{2.30}$$

The above equations express the upwind differencing operation mathematically. We have been unable to find an analytical Fourier representation of these, nor has an empirical approximation suggested itself from observing the power and phase spectra of the simulations.

Chapter 3

Architecture

This chapter describes the high-level structure of the fluid simulation software we implemented to test our ideas, and presents the various visualizations we used to observe the behavior of the simulations.

We began our investigation with a C implementation of Stam's FFT-based fluid solver, as presented in his paper [18]. Simple optimizations such as common subexpression elimination and rearranging the loops for increased cache efficiency improved the simulation performance by a few percent. Restricting grid dimensions to powers of two brings additional optimizations for a total performance increase up to 20%, but most of our experimental measurements in Chapter 4 were made without this extra level of optimization.

Figure 3.1 shows the data flow diagram for Stam's FFT-based simulation. The shaded box contains operations performed in the frequency domain, and the rest is in the spatial domain. The two operations on the border of the shaded box are the forward and reverse Fourier transforms. These are performed in our implementation, as in Stam's, with the popular FFTW library [6].

We modified the order of operations slightly as shown in Figure 3.2 so that new forces are added to the simulation after the advection step is performed, instead of before. We reasoned that since new forces input by the user are not mass-conserving without corrective pre-processing, performing advection on a non-mass-conserving field must reduce accuracy. By performing advection on the mass-conserving output of the previous iteration and then adding new forces just before the next mass conservation operation, physical accuracy is increased without needing to pre-process the input. Stam also made this change in [19]. For the remainder of this report we omit the user input to simplify the diagrams. We conducted our measurement experiments without interactive input.

We changed to a C++ implementation in order to better compare different simulation types and

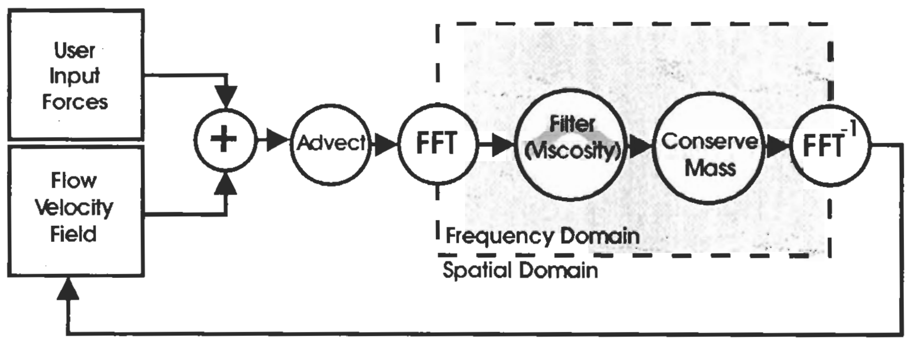

Figure 3.1: Data Flow Diagram of Stam's FFT-Based Solver. New velocities are added to the flow velocity vector field. Self-advection is performed on the result. The simulation data is then transformed to the frequency domain where diffusion and mass conservation take place. The data is then transformed back to the spatial domain for the next iteration.

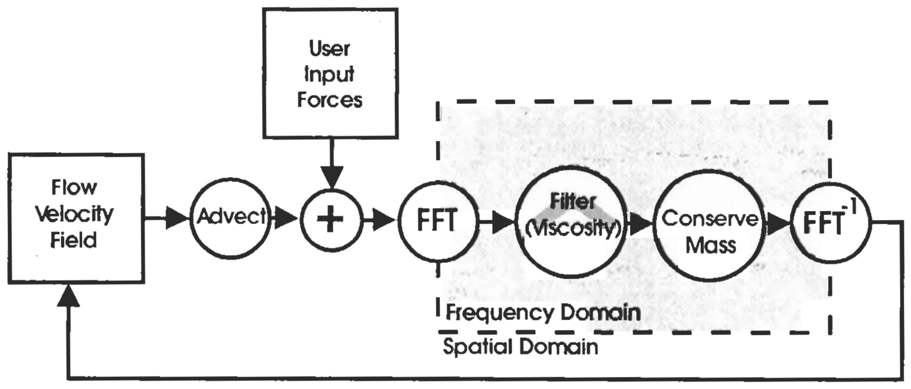

Figure 3.2: Data Flow Corrected to Increase Accuracy. New input is made mass-conserving before advection is applied.

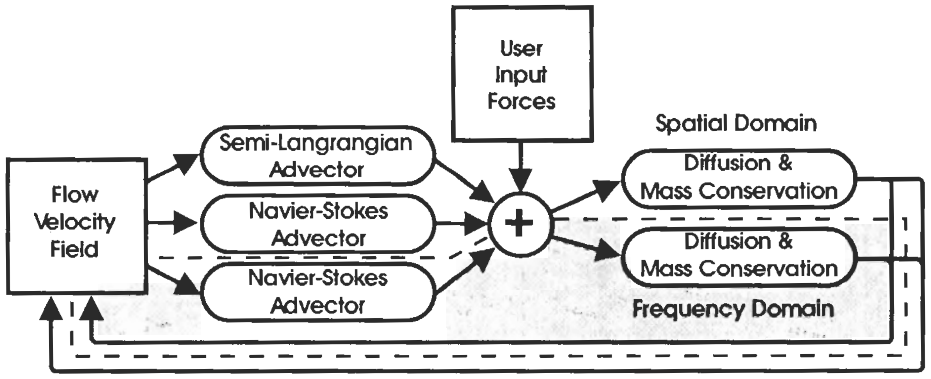

Figure 3.3: Multi-Path Simulation Data Flow. Advection and mass conservation methods are user-selectable, and FFTs are automatically inserted as needed.

try different visualizations with a minimum of code duplication. We implemented a single 2D and a single 3D simulation class each encapsulating multiple advection and mass conservation algorithms that can be selected at runtime. Fourier transforms are performed on an on-demand basis, so they are automatically omitted from simulations that operate entirely in one domain.

Figure 3.3 shows the data flow diagram for this general simulation class. The specific left-to-right path used is selectable at runtime. Variations of Figure 3.3 will be used in Chapter 4 to illustrate specific configurations of interest.

Figure 3.4 shows the general class instance structure of the program when in operation. The main control panel is the tool by which the user selects dimensionality, grid size and simulation type and creates simulation class instances.

Each simulation class instance also has a control panel of its own, by which the user can adjust simulation parameters and select visualizations ("view" windows). Each view is fed data from the simulation class and displays in its own window. Some view windows also have parameters of their own, with a hidable on-screen control panel to let the user control those.

This program structure separates visualization and rendering methods from the flow simulation pipeline, but the simulation code does need to encode the matching between the types and domains of its own data and the types of visualizations that are available. We had intended to work out a more general, automatic way of matching visualizations to data but this was not done due to time constraints and low relevance.

Note that Figure 3.4 does not show the data flow within the program. Ignoring administrative

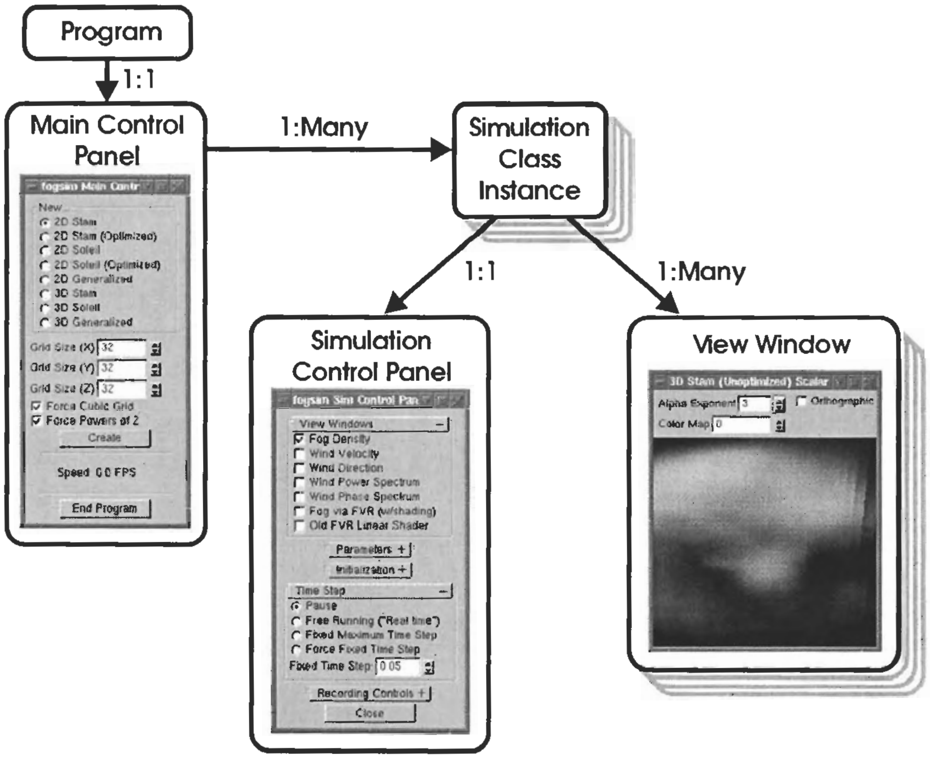

Figure 3.4: Simulation Program Class Structure. The main control panel instantiates simulations. Simulations have their own control panels, and can instantiate multiple view windows.

data, the primary data flow between classes is from the simulation class to its view windows. The second major flow of data is from the view window back to the simulation; when the user interacts with the fluid flow, the input is captured by the view window and passed up to the simulation. The third main communication channel is between the simulation class and its control panel, and is only used when the user interacts with the control panel to change parameters.

We also added an "offline" mode to the program so that movies and timing measurements can be made without having to stay logged in or keep a window open. In offline mode, the control panels are not used. A single simulation instance is created, and any user-specified view windows render their output to image files instead of to the screen. As of this writing not all of the view types support offline rendering, but with the use of an offline OpenGL library like OSMesa they can be modified to do so.

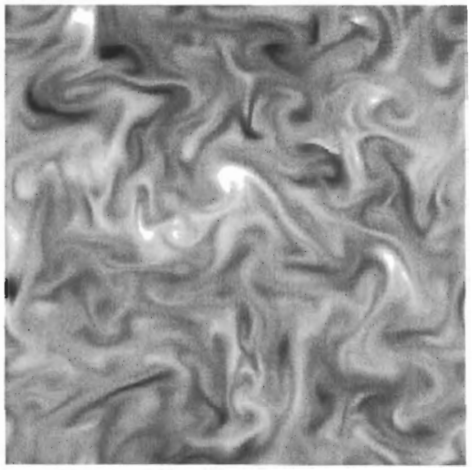

Figure 3.5: 2D Spatial Domain Scalar View. This view displays a single 2D scalar field as a greyscale image.

3.1 Visualizations

The remainder of this chapter presents the various data views that were implemented. There are three axes of classification for the view types: 2D versus 3D, spatial domain versus frequency domain, and scalar versus vector. Saying a view is "scalar" means that it displays a single scalar data field such as fog density. Saying that it is a "vector" view means that it displays vector data or multiple scalar fields at once.

The 2D scalar view shown in Figure 3.5 renders a scalar field such as fog density data as greyscale intensities in a 2D bitmap. The example image shows typical output from a 256^2 simulation, with the fog densities kept in the range [0,1].

The 3D scalar view (Figure 3.6) works similarly to its 2D counterpart, rendering the $n \times n \times n$ fog data to a set of n axis-aligned $n \times n$ textures. The textures are arranged into a cube and displayed using either a perspective or an orthogonal projection. The example images are taken from a 64^3 simulation.

In the 2D version, fog density is mapped directly to pixel value. In the 3D version, a pair of functions are used to map fog density and spatial location to color and opacity for the texture pixels, resulting in the variety of results shown. User-tunable parameters control the mapping functions. At the time of this writing, this view class only supports axis-aligned rendering viewed from the

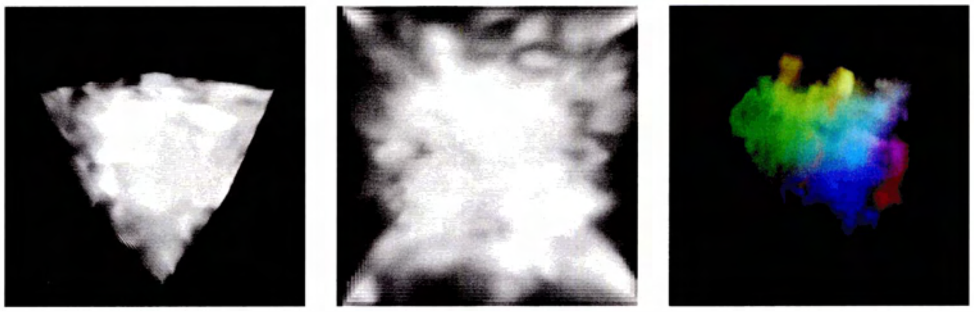

Figure 3.6: 3D Spatial Domain Scalar View using Axis-Aligned Textures. This view extends the 2D version to 3D by arranging RGBA slice images into a cube. Multiple transfer functions can be applied.

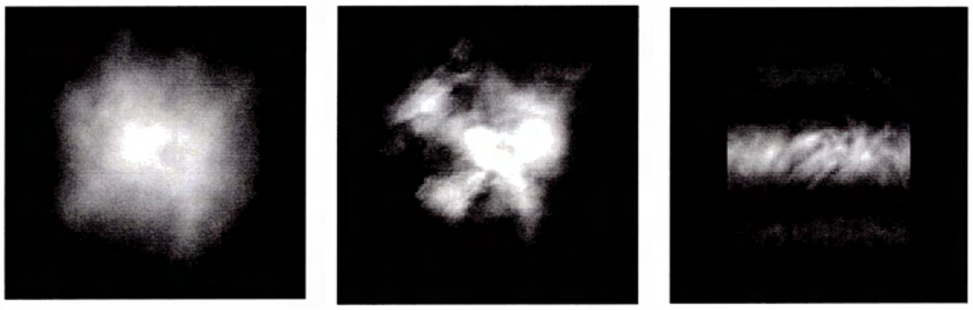

Figure 3.7: 3D Frequency Domain Fog Views via FVR. The left image is normal FVR output. The middle image is linearly shaded, and the right image shows our light beam effect.

negative Z axis. It can be extended to align with the most screen-parallel axes or use fully screen-aligned slicing as in [29], or even to utilize the hardware 3D texturing features of recent graphics processors.

The 3D FVR view (Figure 3.7) implements the Fourier Slice Projection theorem to render fog density data directly from the frequency domain to the screen, or from the spatial domain with a Fourier transform preprocessing step. Linear shading and light beams are options for this view. The code for this view was largely written by Steve Kilthau and Paul Stark [22], and the lighting code by Alireza Entezari [10]. The left image in Figure 3.7 shows the FVR results without lighting effects. The middle image shows an example linear shaded image, and the right image shows a light beam. More example images can be seen in Figure A.3 and Figure A.4 in Appendix A. All three images

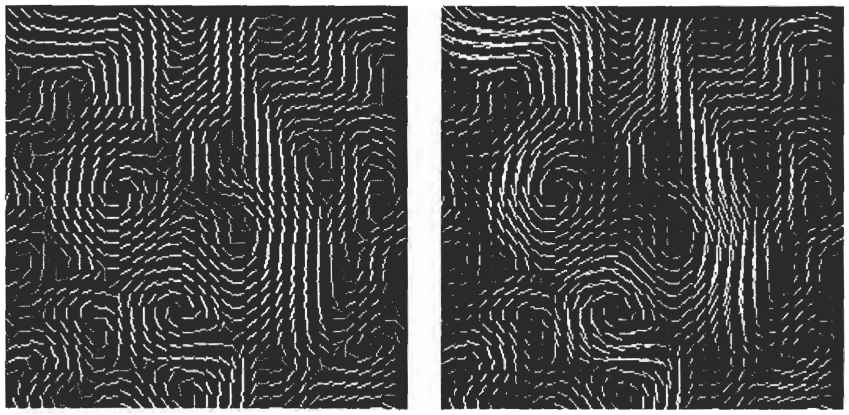

Figure 3.8: 2D Spatial Domain Flow Vector View. In the left image all line lengths are set to one grid unit. In the right image the lengths are set so that the average is one grid unit.

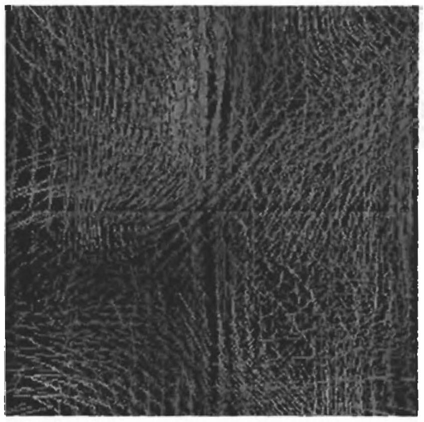

Figure 3.9: 3D Spatial Domain Flow Vector View. The same idea as the 2D version, but displays a cube of vectors instead of a square.

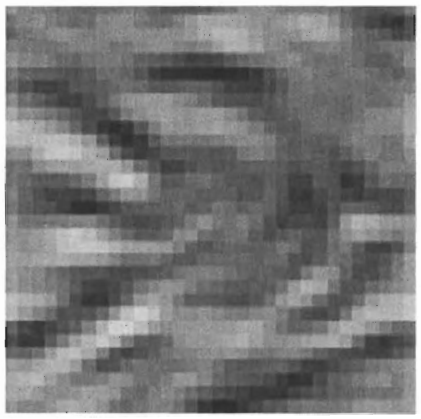

Figure 3.10: Alternate FVR View. This version does not pad the frequency domain data, resulting in a brightness increase due to ghost artifacts overlapping the image.

are from a 64^3 simulation.

Figure 3.10 shows example output from a different implementation of the FVR renderer with linear shading. This version produces brighter-looking images because it does not pad the frequency domain data with zeroes. This results in ghost images that seamlessly overlap, contributing to the brightness without hiding the shading effect.

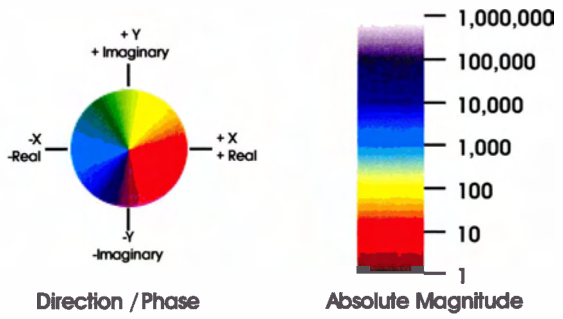

Figure 3.11: Color Maps Used for Direction and Phase (Left) and Magnitude (Right). These color maps are used by the magnitude, phase and direction views

The 2D and 3D vector views (Figure 3.8 and Figure 3.9) display flow direction and magnitude by drawing oriented lines of variable length and color. The lines are rooted at the simulation grid sample points. Line color is set according to velocity relative to the average, with white being the maximum and dark grey the minimum.

Both views have three modes governing the length of the lines: absolute, relative and unit. In absolute mode the line length represents the velocity of the fluid at that sample point in units of the sample grid spacing, giving literal information about the velocities present in the simulation. In relative mode, the line lengths are scaled so that the average velocity present in the simulation has a corresponding line length of one grid unit, giving an impression of the range of velocities present. Finally, in unit mode, all lines are one grid unit in length, allowing all flow directions to be seen regardless of their absolute or relative velocities. In the absolute and relative modes, very small velocities have their direction obscured by the correspondingly short line length assigned to them.

The unit vector mode is shown on the left side of Figure 3.8, and the right side and Figure 3.9 are in relative mode. Absolute mode looks similar to relative mode, but is less interesting to look at when all velocities are large or small. The images shown Figure 3.8 are from a 32^2 simulation and Figure 3.9 shows an example image from a 16^3 simulation. At larger simulation sizes these views become less useful due to crowding and scaling of the lines. It is also difficult to extract any useful information from the 3D version without interactively changing the viewpoint.

The 2D HSV vector view (Figure 3.12) is an experimental visualization that maps flow direction to pixel color in a 2D bitmap. The color mapping is simply direction to hue as shown on the left side of Figure 3.11. Velocity relative to the average can optionally be mapped to value, producing

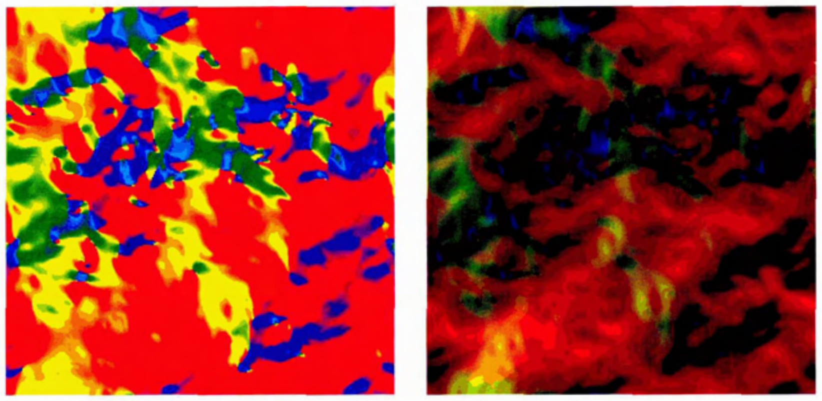

Figure 3.12: 2D Spatial Domain Flow Direction Visualization. Here color indicates flow direction, and brightness optionally indicates relative velocity, as shown in the right image.

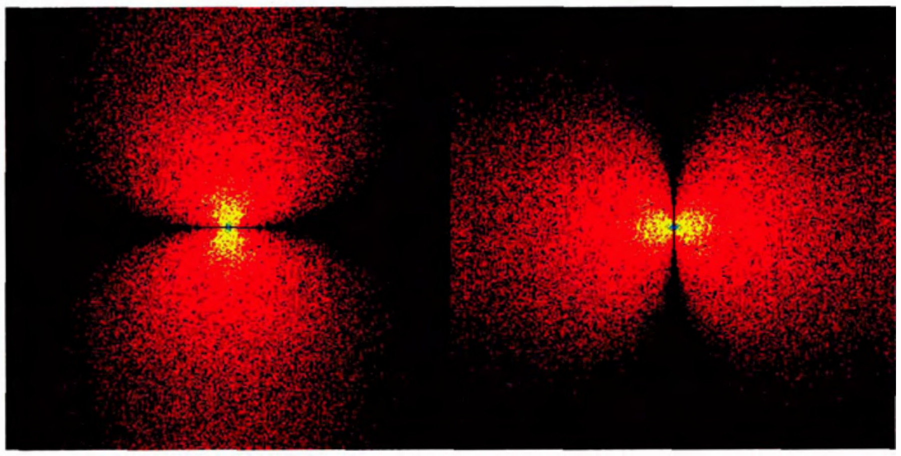

Figure 3.13: 2D Frequency Domain Flow Vector Energy Visualization. This view plots the absolute magnitude of the flow vector U (left) and V (right) components.

images like the right side of the figure. It is not clear whether this visualization has any scientific value, but it does make for attractive displays and overcomes the density limitation of the line-based vector views. These images are from a 256^2 simulation.

The 2D and 3D frequency domain magnitude views, shown in Figure 3.13 and Figure 3.14 respectively, plot the frequency domain representation of the flow vector data. Note that the frequency

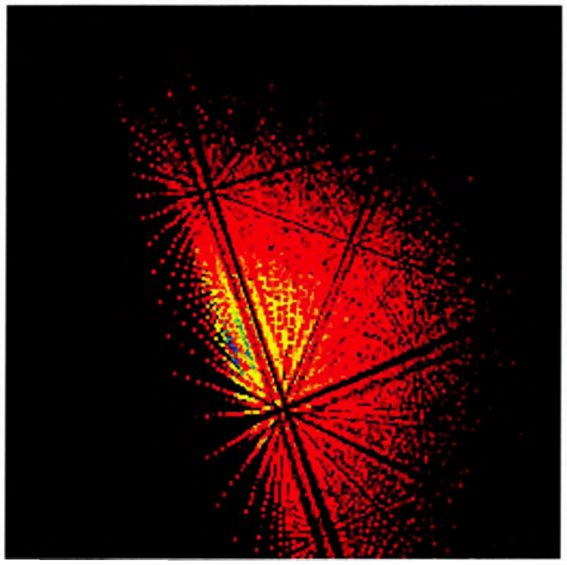

Figure 3.14: 3D Frequency Domain Flow Vector Energy View. This view plots the sum of the magnitudes of the flow vector U, V and W components.

domain representation of all data is complex, even if it is strictly real in the spatial domain.

The 2D version plots the absolute magnitude spectra of the complex $[U, V]$ flow vector components separately, side by side. The left half of Figure 3.13 shows the magnitude plot of the U components of the frequency domain wind data versus ω_x and ω_y, and the right half plots the V magnitudes. This image is from a 256^2 simulation that was recently seeded with random velocities. As the simulation progresses, the high frequencies quickly die out.

Visualizing this kind of information is more difficult in 3D. The 3D view shown in Figure 3.14 plots the magnitude spectrum of the sum of the $[U, V, W]$ wind components. This is an oversimplified view and while it does show the overall distribution of energy with respect to the origin, it obscures per-component differences in the distribution. This image is from a 16^3 simulation and only shows the $\omega_x \geq 0$ half of the spectrum. Since the spatial domain expression of the simulation is strictly real-valued, its Fourier transform has Hermitian symmetry and so only half of it is shown to reduce clutter. Both the 2D and 3D versions plot their magnitude data using the logarithmic color scale shown on the right of Figure 3.11.

The 2D Fourier phase view (Figure 3.15) is the companion to the 2D Fourier power view shown in Figure 3.13. It plots the phase of the complex data using the direction to color mapping on the

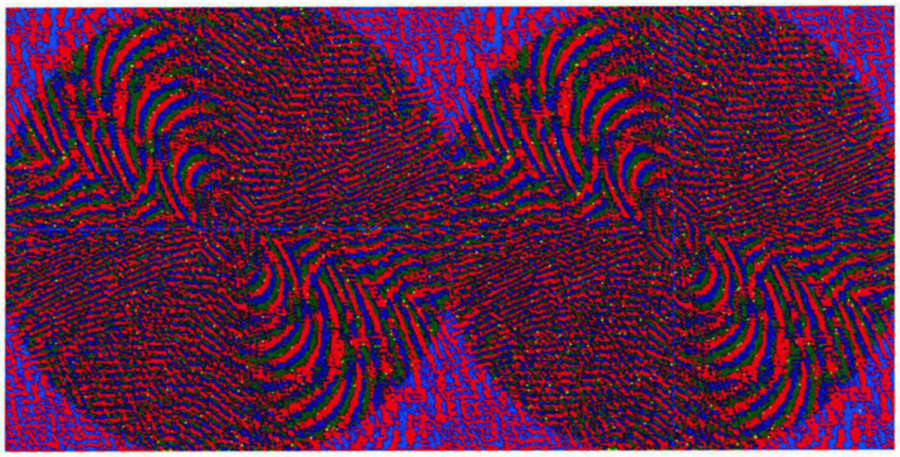

Figure 3.15: 2D Frequency Domain Flow Vector Phase Visualization. This view plots the phase of the flow vector U (left half) and V (right half) components.

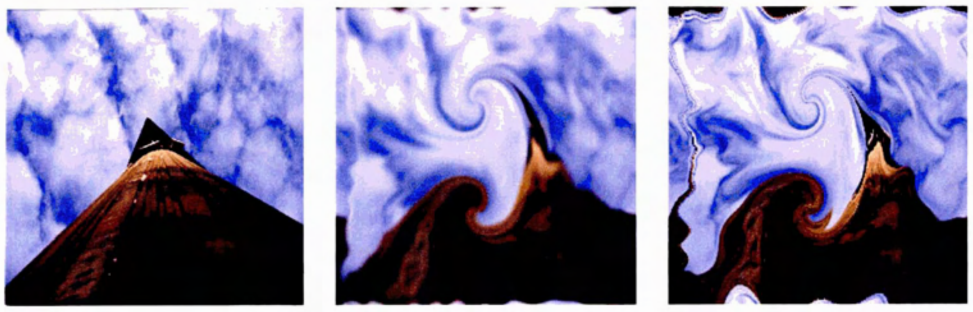

Figure 3.16: 2D Image Warper. The left image is a photograph by the author. The middle image shows the image distorted by a 2D flow using the RGB method described in the text. The right image shows the same distortion using the texture coordinate method.

left of Figure 3.11. This image is from a 256^2 simulation that has lost most of its energy and high frequencies. The phase view did not prove to be very helpful or informative in our investigation. The only properties worth noting are that the size of the circle visible in the figure is inversely related to the fluid viscosity and time step, that everything outside the circle is real-valued, and that moving moire patterns sometimes briefly appear as the simulation settles down into a steady state.

Interesting visual effects can result when the fluid simulation is used to advect things other than fog density. Figure 3.16 shows before and after versions of a picture that has been distorted by the simulation. The leftmost image is the undistorted original image, part of a photograph taken by the author. The other two images are both taken from a 256^2 simulation using the left image as input.

There are three ways a fluid simulation can be used to distort an image (or volume). The naive method is to treat RGB colors from the image as three separate scalar fields and to advect them independently as if they were three fog density fields. This does not work well because the smoothing effect of the simulation will quickly blur out the image and produce color blending. This result is shown in the middle image.

A better approach is to advect two scalar fields, one containing texture X coordinates and the other Y coordinates, which refer to pixel locations in the original image. This way instead of blurring the image, the smoothing effect of the simulation will reduce the area of the original image that is visible and also sometimes create lens effects. The images in the figure were produced using a vector view class designed to interpret its input as either RGB color components or as texture coordinates. The rightmost image shows the result using this method for the same flow that produced the middle image. Although edges and colors from the image are better preserved by this method, there are two drawbacks. One is that aliasing artifacts appear in regions of high distortion, and the other is that interpolation across the periodic boundary of the simulation produces a small mirrored copy of the image that grows over time. This mirror artifact can be seen in the right image of Figure 3.16: the dark spike near the left edge of the image is a narrow mirror image of the tower in the photo.

The mirror artifact can be removed by using the fluid simulation to update a texture *offset* field instead of a texture *coordinate* field. This works as follows: Each display pixel is assigned texture coordinates in the source image. Initially the screen coordinates and texture coordinates of a pixel are equivalent. Over time, the fluid velocity is subtracted from the texture coordinates of the pixels. Essentially the screen pixels become particles that wander across the source image. This removes the mirror artifact because there is no more use of ramp images and therefore no undesired second ramp where the boundary wraps around. These methods were reported in [28] and were used to create special effects for the animated movie *The Prince of Egypt*.

Chapter 4

Results

This chapter presents our performance measurements for the simulations we studied and establishes which is least expensive computationally, and under what conditions. We also discuss our attempt to mitigate the high asymptotic cost of the frequency domain advection expression derived in section 2.3.1.

Figure 4.1 shows the general form of the data flow in our fluid simulation pipeline. We have implemented a generalized form of this simulation pipeline that allows runtime selection of the algorithms used to perform advection, viscosity, mass conservation and rendering. Figure 4.2 shows a data flow diagram of this implementation.

Any path from left to right through this chart constitutes a functional fluid simulation. The shaded areas represent the frequency domain, and the unshaded area represents the spatial domain. A path segment that crosses into a shaded area requires a Fourier transformation of the simulation data, and a path segment that leaves a shaded area requires an inverse Fourier transform. No

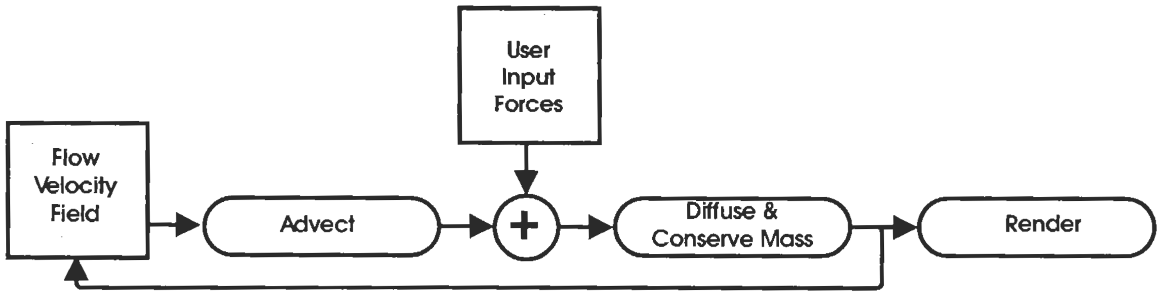

Figure 4.1: General Form of Our Simulation Pipeline. The main loop runs left to right through the advection and mass conservation algorithms.

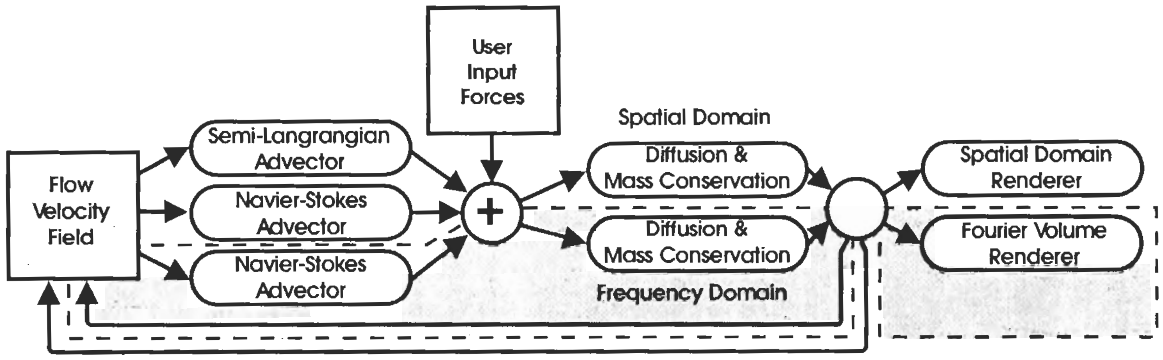

Figure 4.2: All Possible Algorithm Combinations. The choice of advector and mass conserver are user-selectable.

transformations are required between regions of the same color. There are two Navier-Stokes (Eulerian) advection operators shown; one is a spatial domain version that we implemented to verify the algorithm before implementing the other, the frequency domain version.

Figure 4.3 shows three relevant paths through the generalized pipeline. The top path represents Stam's FFT-based domain simulation from [18]. The middle one is equivalent to Stam's spatial domain simulation from [19]. The bottom path is our new frequency domain simulation pipeline. Only the first of these three requires Fourier transforms.

In the following subsection we present our measurements of the relative cost of each of the advection and mass conservation algorithms shown in Figure 4.2 and of the three major paths through the pipeline shown in Figure 4.3. We will then identify the least computationally expensive pipeline for practical application on today's computers.

Several other pipeline configurations are also possible, but we do not discuss them. They are less interesting because they do not present any advantages over the ones we do discuss. We only briefly discuss rendering times because of the huge variety of possible display methods. We omit rendering time from the performance measurements of the various simulation pipelines.

4.1 Relative Cost of Operations

In this section we present our measured processing times for the various methods of advection and mass conservation shown in Figure 4.2.

Our test simulation used a three-dimensional flow vector field to advect itself plus a three-dimensional scalar field representing fog density, requiring a total of twelve scalar convolutions

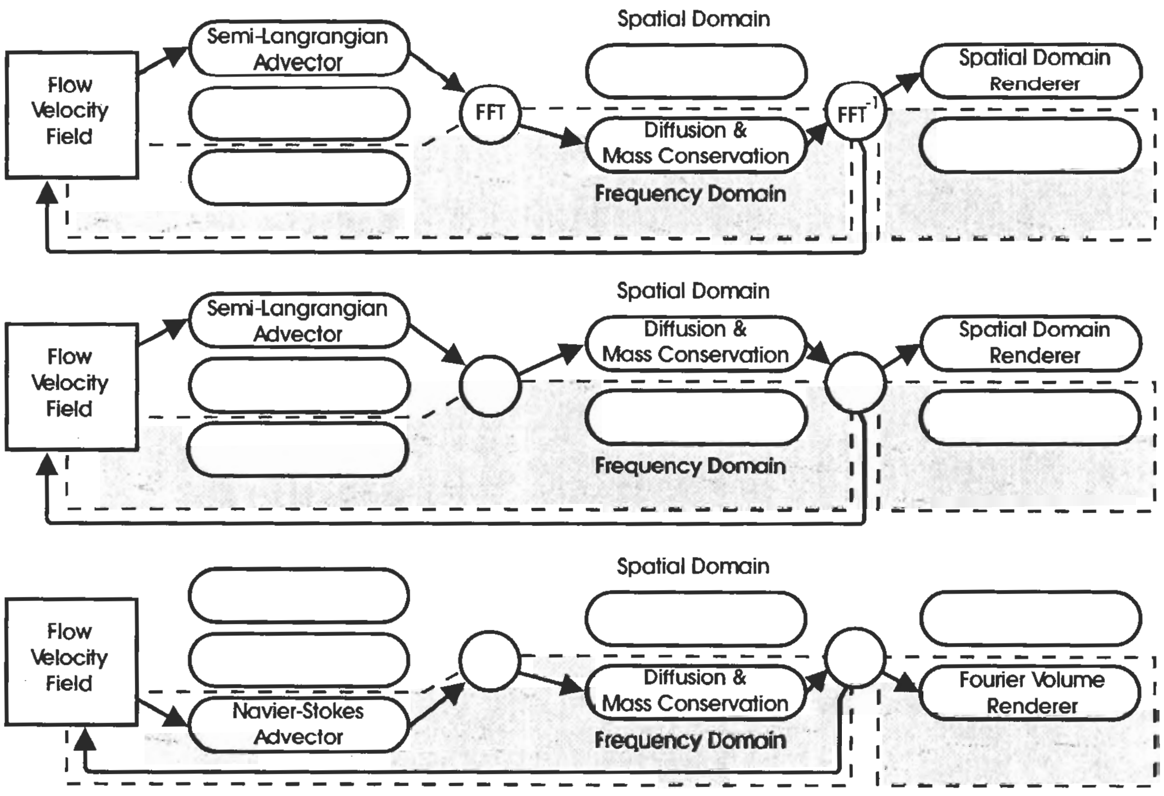

Figure 4.3: Interesting Pipelines Including Fourier Transforms. These specific paths through the general pipeline represent two pipelines presented by Stam and our frequency domain pipeline.

for the frequency domain version. The mass conservation operation was only applied to the flow vector field.

To compare the individual pipeline components on level ground we ignore Fourier transform costs and compare the algorithms in a consistent computing environment. All timing runs were started with randomized velocities and fog density fields. The same data type and memory arrangement was used for all simulation data. All measurements were performed on a 2.4GHz Intel Pentium 4 machine with a 512KB cache and 1GB of RDRAM.

Figure 4.4 shows processing time versus grid size for the advection algorithms in 3D. The horizontal axis is the side length of the cubic simulation grid; 32 means a 32^3 simulation. The vertical axis is logarithmic in units of microseconds; 6 means 10^6 or 1,000,000 microseconds (one second) per iteration. Observe that the cost of the two spatial domain advection algorithms is almost the same, while the cost of the frequency domain version is much greater.

Figure 4.5 presents processing time versus grid size for the mass conservation algorithms in 3D.

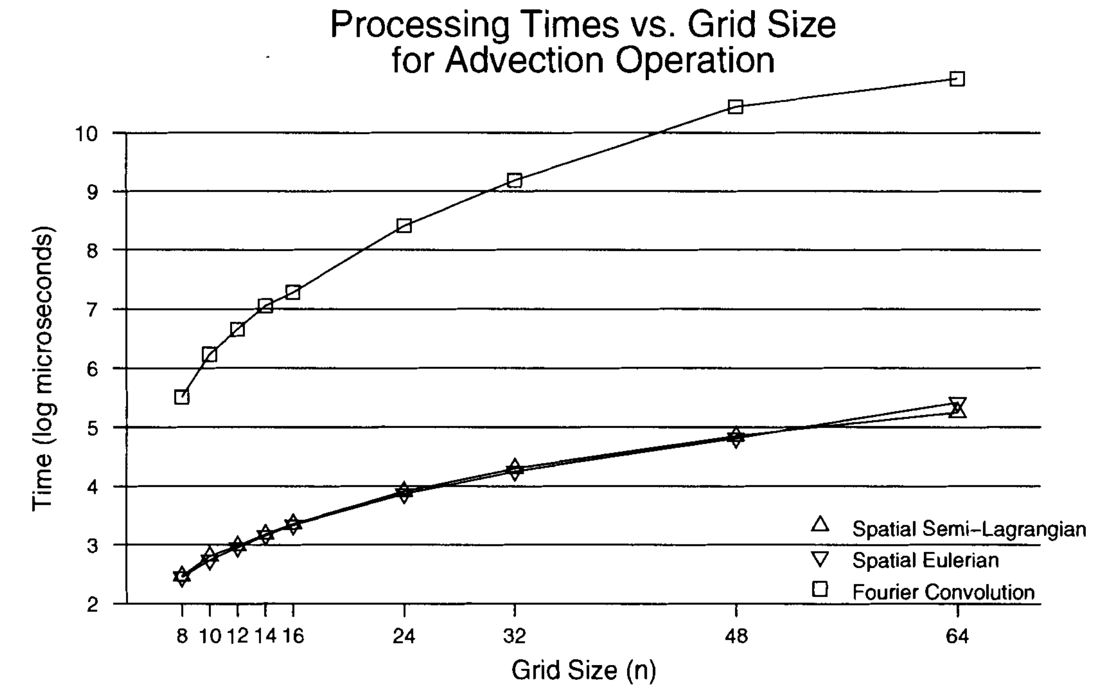

Figure 4.4: Advection Cost vs. Simulation Size. The horizontal axis represents the number of grid points along each side of the 3D simulation grid. The vertical axis plots processing time in microseconds on a logarithmic scale for the three advection algorithms we implemented.

Both of these algorithms are as presented by Stam [18, 19] and perform the diffusion operation at the same time as the mass conservation. Observe that the frequency domain version is consistently an order of magnitude faster. We attribute this to the repeated iteration needed to achieve convergence in the spatial domain version's Gauss-Seidel relaxation solver; Stam used twenty iterations of the solver per simulation step, which is why the slopes of the two curves are the same. If the Gauss-Seidel solver were allowed to converge normally, it would be asymptotically more expensive than switching to the frequency domain.

A multi-grid solver might provide both increased speed and accuracy since it requires $O(1)$ iterations. However, a multi-grid solution involves fairly complex computation whereas the frequency domain solution Stam presented is a simple projection operation. By limiting the number of Gauss-Seidel iterations to twenty, Stam also made the spatial domain mass conservation solver into an $O(1)$ solution, though not perfectly accurate. How much of an improvement a multi-grid solver may be over twenty Gauss-Seidel iterations and how it compares with the frequency domain mass

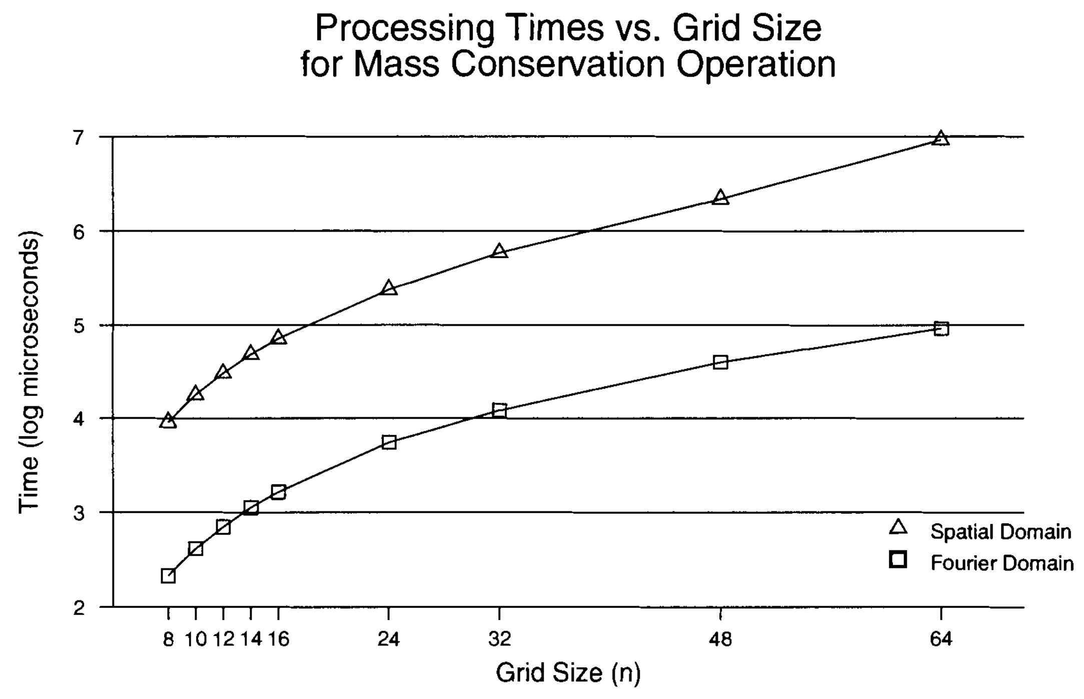

Figure 4.5: Mass Conservation Cost vs. Simulation Size. The horizontal axis is grid side length, and the vertical axis represents the processing time for the two mass conservation methods we implemented.

conservation are left for future investigation.

There are far too many spatial domain rendering methods for us to generally compare them against FVR. At best we can say that any spatial domain renderer must be $\Omega(n^3)$ while FVR is $O(n^2 \log n)$. For a rough example, Figure 4.6 shows the performance of our FVR implementation fed with frequency-domain data compared with software rendering of axis-aligned textured slices fed from the spatial domain. This comparison levels the playing field by removing the hardware acceleration advantage that spatial domain methods currently have access to. It is certainly possible to implement FVR in hardware to gain an equivalent speedup. The important observation here is that the FVR rendering times do not grow as fast as the spatial domain times.

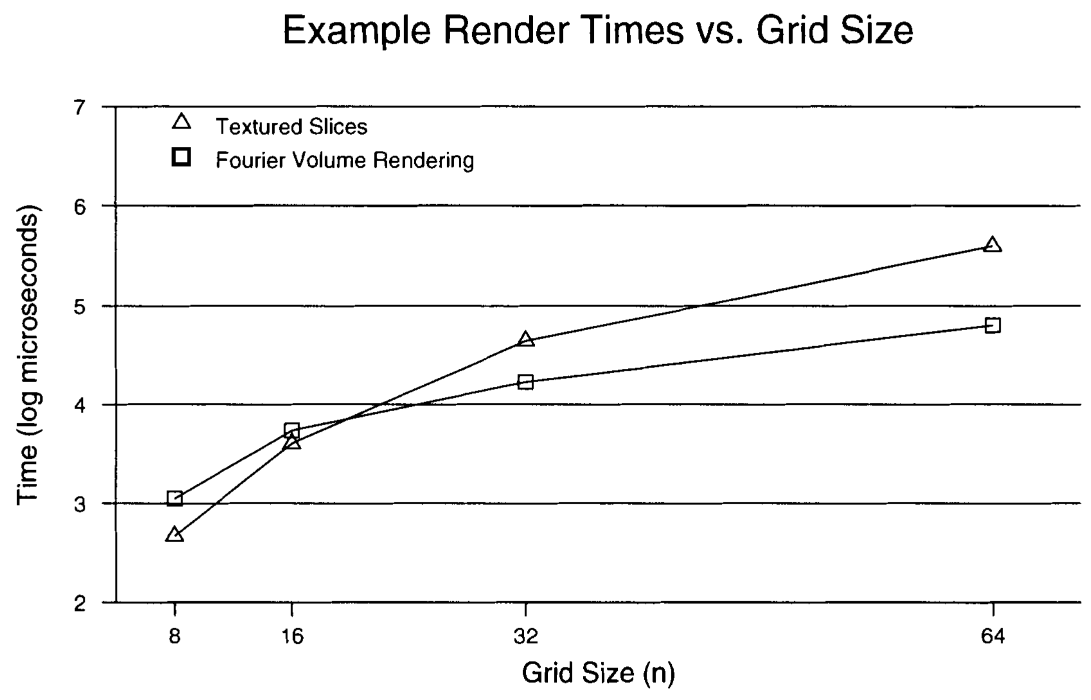

Figure 4.6: Example of FVR Rendering Advantage. This graph shows that FVR rendering time as a function of grid side length grows asymptotically more slowly than an example spatial domain method. The spatial domain method compared against is the 3D textured slice view shown in Figure 3.6 without hardware acceleration.

4.2 Reducing the Frequency Domain Cost

The frequency domain advection operation is by far the most expensive part of our desired simulation pipeline. Convolution is so computationally intensive that it overshadows everything else and destroys any chance of gaining an advantage from the reduced rendering cost of FVR. This section discusses the results of our attempt to reduce the computational cost of the frequency-domain advection step.

We observe that the majority of the energy in fluid simulations is in the lower frequencies (see Figure 3.13). Therefore a logical experiment to try is to reduce the size of the convolution kernel to exclude the large region that is near zero. Of course, high frequencies are present when there is rapid activity and fine detail present in the simulation. To avoid losing this detail, the kernel size might be set adaptively according to the energy distribution in the simulation.

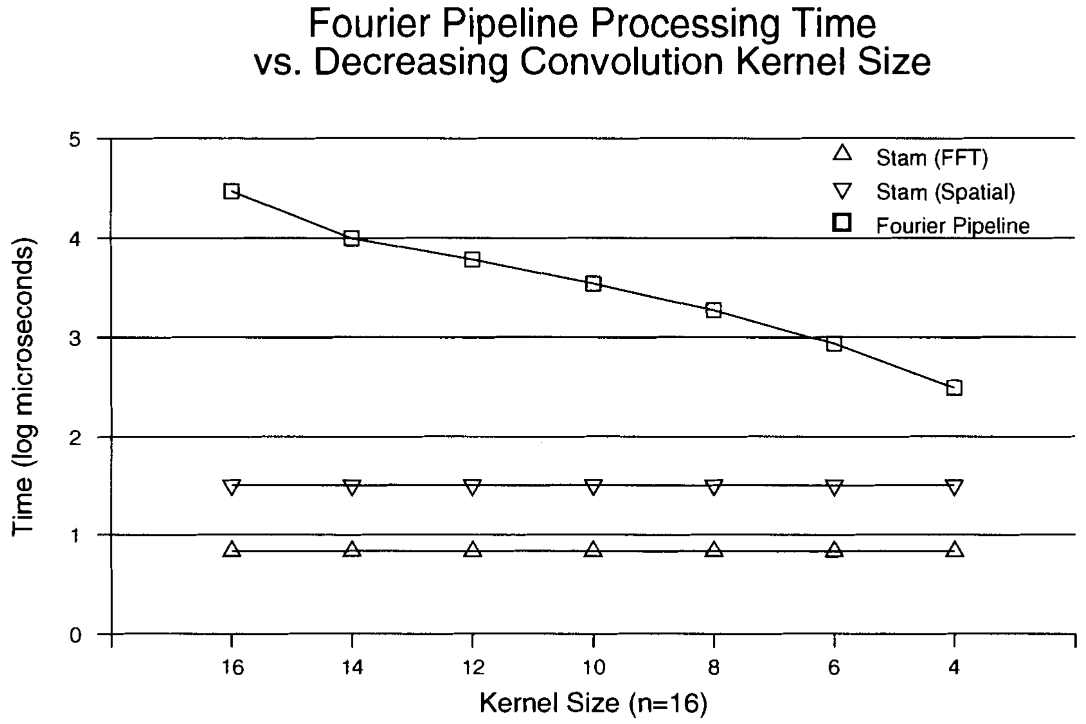

Figure 4.7: Effect of Reducing Convolution Kernel Size. The sloping line represents processing time for a 16^3 frequency domain simulation as the convolution kernel size is reduced from 16^3 (left) to 4^3 (right). For reference, the other two lines show the processing times for Stam's two simulations.

Figure 4.7 shows example computation time for a 16^3 frequency domain simulation as the kernel size is reduced from 16^3 down to 4^3. These measurements were taken from an optimized version of the software.

The convolution cost does drop quickly as the kernel size is reduced, but it still remains higher than the cost of the spatial domain operations. It is impossible to reduce the asymptotic cost of the convolution below the cost of the spatial domain operations by reducing the kernel size, because n^3 data points still need to be visited by the algorithm and because reducing the kernel size too much results in loss of quality and stability.

Although it is difficult to quantify, we make some casual observations about the effects of kernel size reduction on the behavior of the flow simulation. We observed that when the kernel size was in the vicinity of half the simulation size the flows started behaving oddly and unnaturally, exhibiting spontaneous motion and lack of proper advection. At the smaller kernel sizes, all semblance of flow

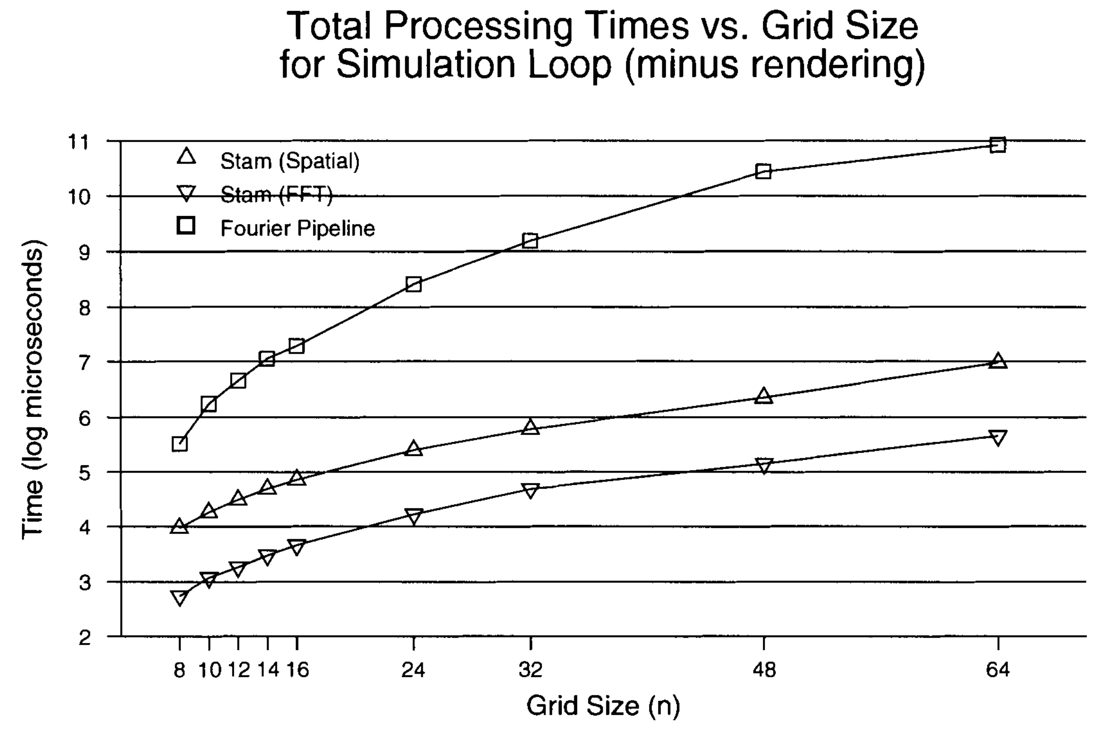

Figure 4.8: Total Cost of Three Interesting Simulation Pipelines. This graph plots the total simulation times for the three pipelines identified in Figure 4.3, not counting rendering.

was lost and the simulation typically became either static or grossly unstable.

In cases where physical accuracy is not absolutely required, it might be possible to combine a small frequency domain simulation with the frequency synthesis method of Stam and Fiume [20] to create the illusion of finer detail.

4.3 Relative Cost of Simulations

We present the total cost, including Fourier transforms, of the simulation pipelines from Figure 4.3. We use the FFTW library version 2.1.3 for the Fourier transformations [6]. Figure 4.8 presents our measured total cost for each of the three pipelines. We ignore rendering time here.

Again the frequency domain version is much more expensive than the others because of the high cost of convolution. Also observe that there is an order of magnitude difference between the purely spatial domain pipeline and the FFT-based pipeline, owing to the difference in the cost of their mass conservation algorithms. The repeated iteration required by the spatial domain version outweighs the cost of the Fourier transforms required by the frequency domain version. This will also be shown in the experimental analysis below.

Referring to Figure 4.8, we now derive expressions for the run time of each pipeline and show the tradeoffs between them. We will discard asymptotic notation here and include the various constants calculated from the actual measured run times.

In this analysis we use the letter A to represent advection costs, C to represent mass conservation costs, and F to represent the Fourier transform costs. For the advection operators, let $A_s n^3$ represent the run time of the spatial domain semi-Lagrangian version, and $A_f k n^6$ represent the run time of our frequency domain version, where $0 < k \leq 1$ sets the convolution kernel size relative to n. Similarly let $C_s n^3$ be the cost of mass conservation in the spatial domain, and $C_f n^3$ in the frequency domain. Finally, let $F n^3 \log n$ be the combined cost of the forward and reverse Fourier transforms. The expressions for the total run times of these three pipelines are then:

$$T_{\text{spatial}} = (A_s + C_s) n^3 \tag{4.1}$$

$$T_{\text{fft}} = (A_s + C_f + F \log n) n^3 \tag{4.2}$$

$$T_{\text{fourier}} = (A_f k n^3 + C_f) n^3 \tag{4.3}$$

From this we can see that the spatial domain pipeline (Equation 4.1) is faster than the FFT-based pipeline (Equation 4.2) when $C_s < C_f + F \log n$. The FFT-based pipeline (Equation 4.2) is faster than the frequency domain pipeline (Equation 4.3) when $A_s + F \log n < A_f k n^3$ which is true for all but the smallest n or k. More precisely, it is false only when $kn^3 \leq \frac{A_s + F \log n}{A_f}$. Satisfying this condition requires either unreasonable small values of n (on the order of 1, which isn't much of a simulation grid), or very small values of k, which makes the kernel size tiny in comparison to the simulation size.

Determining the acceptable reduction in k, including the criteria for acceptability, and whether or not the convolution kernel size kn^3 can be replaced with a constant size kernel, are avenues of future research.

From our experimental measurements we estimate the actual values of the above constants to be as follows. These constants are for time units of microseconds on our 2.4GHz test machine with $k = 1$.

$$\begin{aligned}
A_s &= 0.65, \\
A_f &= 1.485, \\
C_s &= 22, \\
C_f &= 0.383, \\
F &= 0.0208
\end{aligned}$$

Rewriting the previous set of equations using these constants, we have:

$$T_{\text{spatial}} = 22.65n^3 \tag{4.4}$$

$$T_{\text{fft}} = (1.033 + 0.0208\log n)n^3 \tag{4.5}$$

$$T_{\text{fourier}} = (1.485kn^3 + 0.383)n^3 \tag{4.6}$$

It is obvious that the fully frequency domain pipeline ($T_{fourier}$) has the worst asymptotic profile, and that the FFT-based pipeline (T_{fft}) is asymptotically more expensive than the fully spatial domain pipeline ($T_{spatial}$). We will now derive expressions for the tradeoff points, or values of n where the identity of the lowest cost simulation changes. For our test conditions, the frequency domain pipeline is least expensive when

$$\begin{array}{rcl}
T_{\text{fourier}} & < & MIN(T_{\text{spatial}}, T_{\text{fft}}) \\
(1.485kn^3 + 0.383)n^3 & < & MIN(22.65n^3, (1.033 + 0.0208\log n)n^3) \\
1.485kn^3 + 0.383 & < & MIN(22.65, 1.033 + 0.0208\log n) \\
kn^3 & < & \dfrac{MIN(22.267, 0.65 + 0.0208\log n)}{1.485} \\
kn^3 < 14.995 & \text{and} & kn^3 < 0.4377 + 0.014\log n \\
\text{If we take} & k = 1 & \text{(full convolution), then} \\
n^3 < 14.995 & \text{and} & n^3 < 0.4377 + 0.014\log n
\end{array}$$

For the first condition to be satisfied, n must be less than 2.47. For the second condition, n must be less than unity. These are uselessly small simulation sizes. The convolution scale factor k can be used to increase these minimum values of n, but in order to reach any meaningful simulation size k must be very small, which means sacrificing detail, accuracy and stability. To reach a grid size of

$n = 2$ while preserving the frequency domain simulation as the fastest, we must set $k <= 0.0427$. Since k controls the size of the convolution kernel in terms of the simulation size, this means we would be trying to convolve with a kernel much smaller than the grid sample spacing, which is not a convolution at all. The situation only gets worse as we try to achieve larger values of n.

We conclude from this that for the type of computers available to us at the time of this writing, there are no conditions under which the fully frequency domain pipeline is both the fastest and useful. Perhaps the advent of quantum or biological computing will change this outcome.

We will now make a comparison between Stam's FFT-based and fully spatial domain pipelines, and identify the overall least expensive pipeline given our test conditions. The FFT-based pipeline is asymptotically more expensive than the spatial domain version. From our measurements, the spatial domain pipeline becomes the fastest when

$$\begin{aligned} T_{\text{fft}} &> T_{\text{spatial}} \\ (1.033 + 0.0208 \log n)n^3 &> 22.65n^3 \\ 1.033 + 0.0208 \log n &> 22.65 \\ \log n &> \frac{22.65 - 1.033}{0.0208} \\ \log n &> 1039.28 \\ n &> 2^{1039} \end{aligned}$$

A simulation of that size has $(2^{1039})^3 \approx 2.04 \times 10^{938}$ sample points, which is far greater than the estimated number of atoms in the observable universe (in the vicinity of 10^{80}) and therefore beyond our present ability to store in a computer. From this we conclude that the FFT-based pipeline is the most practical of the three for application on current computers.

The fully spatial domain pipeline might be made more attractive by replacing the Gauss-Seidel solver with another sparse system solver such as multi-grid or successive over-relaxation, but we have not evaluated these options. We note that Stam also made this observation [17] but decided to use Gauss-Seidel with a constant number of iterations based upon the claims of Foster and Metaxas [5].

It is worth noting that the pipeline that implements the FFT-based simulation with semi-Lagrangian advection does not represent the least expensive possible path. The gradient operator requires fewer machine instructions to compute than linear interpolation. Our spatial domain Eulerian implementation of the traditional Navier-Stokes advection term has $A_s = 0.562$ versus $A_s = 0.65$ for the spatial

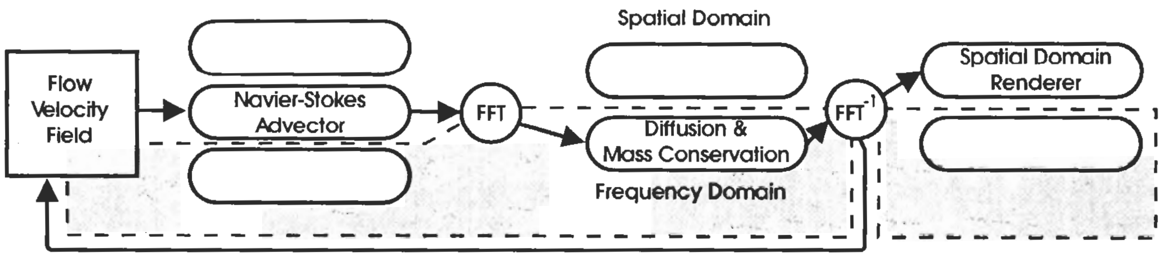

Figure 4.9: Least Expensive Flow Simulation Pipeline. Of the methods we implemented, this simulation pipeline configuration has the lowest total processing time.

domain semi-Lagrangian version, so the pipeline shown in Figure 4.9 is the fastest for all n. If the grid size is restricted to powers of two to gain some code optimizations, this difference increases slightly to $A_s = 0.25$ for the Eulerian versus $A_s = 0.4$ for the semi-Lagrangian version. The tradeoff is that the Eulerian form is unstable. In order to stabilize it, the time step must be limited by the CFL condition shown in Equation 2.26 and upwind differencing must be employed.

The time step limitation of the Eulerian form is a major obstruction for real-time applications such as video games. Conversely, the reduced physical accuracy and increased dissipation of the semi-Lagrangian form are undesirable for scientific applications where prediction of real physical results is important.

Another tradeoff when choosing a simulation type is ease of implementation. Although the FFT-based pipeline is fastest, it does depend on having an efficient implementation of the Fast Fourier Transform. This increases the overall complexity of the simulation program and also increases the code size of the final executable. The latter may make its use less attractive on limited platforms such as video game consoles. The spatial domain pipeline, while slightly slower, is easier to implement and requires fewer lines of code.

The overall conclusion from our experimental results is that the frequency domain simulation is not useful when speed is an issue, and the choice between the other simulation types depends upon the intended application.

Chapter 5

Conclusions

Our motivation for creating a frequency domain fluid simulation pipeline was to take advantage of the reduced rendering cost offered by Fourier Volume Rendering. An obvious drawback of the frequency domain simulation we have presented is its increased computational cost. Experiments and observations lead us to believe that although we can reduce the cost of the frequency domain advection algorithm, it is unlikely that a good tradeoff can be found between simulation speed and other desirable properties such as visual realism and numerical stability. A thorough analysis is left for future work.

It is doubtful that it will be possible to make a physically accurate frequency domain simulation faster than a partially or fully spatial domain solution such as Stam's. Approximate or synthetic solutions may have a better chance, but realism and interactivity will present challenges to those approaches.

Simulation stability is an unsolved problem in the frequency domain. Eulerian Navier-Stokes solvers are unstable. There are various well-known spatial domain methods that stabilize them, such as using small time steps, Runge-Kutta stepping, and upwind methods. We found upwind methods combined with the CFL condition to be very effective, but we have not yet found a satisfactory way of stabilizing the frequency domain simulation. We consider the use of small time steps or intelligent stepping schemes unsatisfactory for real-time applications because of the extra computation required.

It may be possible to make a fully spatial-domain pipeline based on Stam's [19] as fast as the FFT-based version by replacing the Gauss-Seidel solver with something more efficient. Future work in this direction includes an investigation of the use of successive over-relaxation (SOR) or multigrid methods for the solution of the diffusion and mass conservation operations.

The remaining conclusions are about Fourier Volume Rendering and the challenges inherent in making it attractive for video game or film applications. The FVR technique produces X-ray images of the data, without occlusion. Interactive applications require the fluid simulation to react to objects passing through the fluid - for example, an actor moving through a fog bank, or a video game character running through smoke to escape a fire. FVR produces a single 2D image directly from the 3D simulation, making it difficult to create the impression that a character is embedded within the simulation; there is no occlusion in an X-ray projection.

We believe that occlusion of the fog or smoke behind an object can be achieved by convolving with a low-resolution mask that effectively zeroes out the occluded region. This is similar in concept to the light beams we present in Appendix A, except that the goal is to darken the occluded region instead of lightening it. One occlusion method for simple scenes involving one or a few objects embedded in the fog is be to split the simulation into foreground and background parts and render them separately.

Another challenge arises when we want moving objects to add new forces to the simulation. Since video game characters and other simulated objects are typically represented in the spatial domain, interaction is difficult. One approach is to voxelize the spatial domain shape of a character and transform it into the frequency domain. Motion can be represented by phase-shifting the frequency domain representation, but shape changes require re-voxelizing and re-transforming. Another potential solution is to calculate the new forces on a coarse scale in the spatial domain, transform them to the frequency domain and add the results to the simulation.

Shading bears more future investigation, to simulate the bright edges of clouds on the light-facing sides. We have already had some success with this by using simple linear shading as shown in appendix A, but that requires gradient magnitudes to be calculated in the spatial domain. It may be possible to approximate this calculation in the frequency domain. Another possibility for shading is the spherical harmonic method developed by Entezari et al [2]. This method requires updating nine volumes instead of just one, but that might not be a major problem on parallel architectures.

5.1 Contribution

The main contributions of this thesis are the derivation and analysis of a frequency domain advection operation for fluid flows, and the resulting conclusion that physically realistic flow simulation in the frequency domain is very computationally expensive.

We believe the light beams presented in Appendix A are also new.

Appendix A

Lighting in the Fourier Domain

This appendix presents two lighting effects that we created in an attempt to make Fourier Volume Rendering (FVR) more attractive for use with fluid simulations. We discuss linear shading, which requires some data from the spatial domain, and we introduce a fully frequency-domain method of generating cylindrical light beams without modifying the source data.

This discussion of frequency domain lighting effects is presented in an appendix because it is not substantially the author's own work. The basic Fourier Volume Rendering code was developed by Paul Stark and Steve Kilthau to support Paul Stark's work [22], and the lighting extensions were mostly derived by Alireza Entezari [10] and were also implemented by him to produce the images.

Fourier Volume Rendering is based on the Fourier Slice-Projection theorem. The relevant interpretation of the theorem is that any 2D slice through the origin of a 3D frequency volume will contain the spectrum of an X-ray projection of the 3D spatial domain version of the data. In other words, in order to render an X-ray image directly from a 3D frequency domain simulation, we only need to extract the appropriate 2D slice from the 3D volume and apply a 2D inverse Fourier transform to it.

Various lighting effects can be achieved efficiently in the frequency domain; this is mainly due to the fact that the lighting effects need only be computed along the slice that is being taken in the frequency domain. The basic cost of the lighting operations is $O(n^2)$ as opposed to $O(n^3)$ for globally applied techniques.

One lighting effect that can be highly efficient is a *light beam.* We can simulate a cylindrical beam with a higher ambient intensity in the middle, shining through a medium such as fog or smoke. The light beam in the spatial domain is a constant function along the beam direction and a box function in the directions perpendicular to the beam direction. Figure A.1 shows a 2D example of a

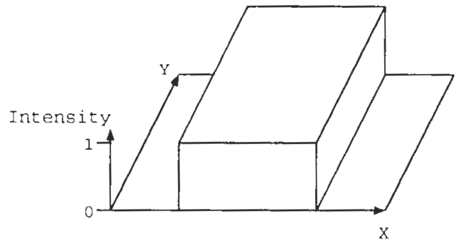

Figure A.1: 1D Box Filter Swept to Form a 2D Beam. This is a conceptual illustration of how a box function can be projected through a higher dimension to produce a beam. Our light beams are formed by projecting a 2D box function through 3-space.

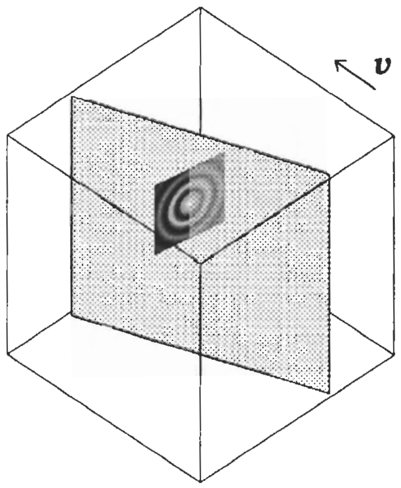

Figure A.2: FVR Slice Convolved with 2D Sinc Filter. The first step of rendering from frequency domain data involves extracting a 2D slice. While calculating the slice values, we convolve with a small 2D sinc image to produce light beams.

beam shining along the Y-axis.

The discrete Fourier transform of the light beam is a Dirac delta function along the light beam direction and a 2D sinc function on the plane perpendicular to the beam direction. During the slicing step (represented by the shaded plane in Figure A.2), we convolve with a 2D function in the frequency domain, represented by the small sinc image embedded in the slicing plane. For instance if the light beam in the spatial domain is aligned on the vector $\vec{v}$ (the arrow in the figure), then its representation in frequency domain is a 2D sinc perpendicular to $\vec{v}$:

$$F(\vec{\omega}) = \begin{cases} \mathrm{sinc}(\sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}) & \text{if } \vec{v} \cdot \vec{\omega} = 0 \\ 0, & \text{otherwise} \end{cases}$$

The condition $\vec{v} \cdot \vec{\omega} = 0$ identifies the $\vec{\omega}$ points that lie on the plane that is perpendicular to $\vec{v}$ and through the origin. The sinc function on that plane is the Fourier transform of a light beam with a circular cross section. A beam with a square cross section can be easily obtained by replacing the $\mathrm{sinc}(\sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2})$ term with the separable sinc function on the plane. For the 2D circular

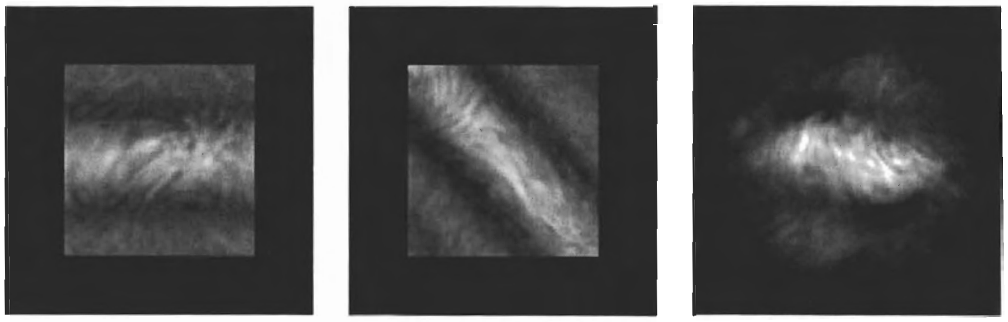

Figure A.3: Examples of Cylindrical Light Beams with Varying Orientation. These images were produced using our light beam extension of FVR. The left two images show different beam directions with an axis-aligned view. The rightmost image shows the same beam as the middle image from a different view direction.

sinc kernel we managed to get convincing results with a relatively small kernel size of 5×5. We approximated the sinc function with a Gaussian function to avoid the ringing effect.

Because the convolution does not need to be evaluated throughout the entire 3D volume of frequency data, but only on the plane that we are slicing from the frequency data, simulating the light beam is a 2D convolution with a small kernel. The images in Figure A.3 show our cylindrical light beams shining through a 64^3 fog simulation at various angles.

Another important lighting effect that can be effectively achieved with the frequency domain data is *diffuse lighting*. In order to compute the light contribution at any point in the data, we need to know the cosine of the angle between the gradient with the light direction. Computing the gradient of the data in frequency domain is accurate and simple as illustrated in Equation A.1. Here we use d to represent the fog density scalar field in the spatial domain, and D for its frequency domain equivalent.

$$\mathcal{F}\{\nabla d(\vec{x})\} = \vec{\omega} D(\vec{\omega}) \tag{A.1}$$

The more the light is aligned with the gradient, the more the contribution at that point is. However, we want to avoid the *negative lighting* effect for the areas that are facing away the light vector. Therefore, the light contribution is proportional to $\max(0, \nabla d(\vec{x}) \cdot \vec{L})$ where $\vec{L}$ denotes the light direction.

The major challenge of achieving the diffuse lighting effect in frequency domain is the non-linearity of the $\max$ function as it does not have a nice Fourier transform. As mentioned earlier

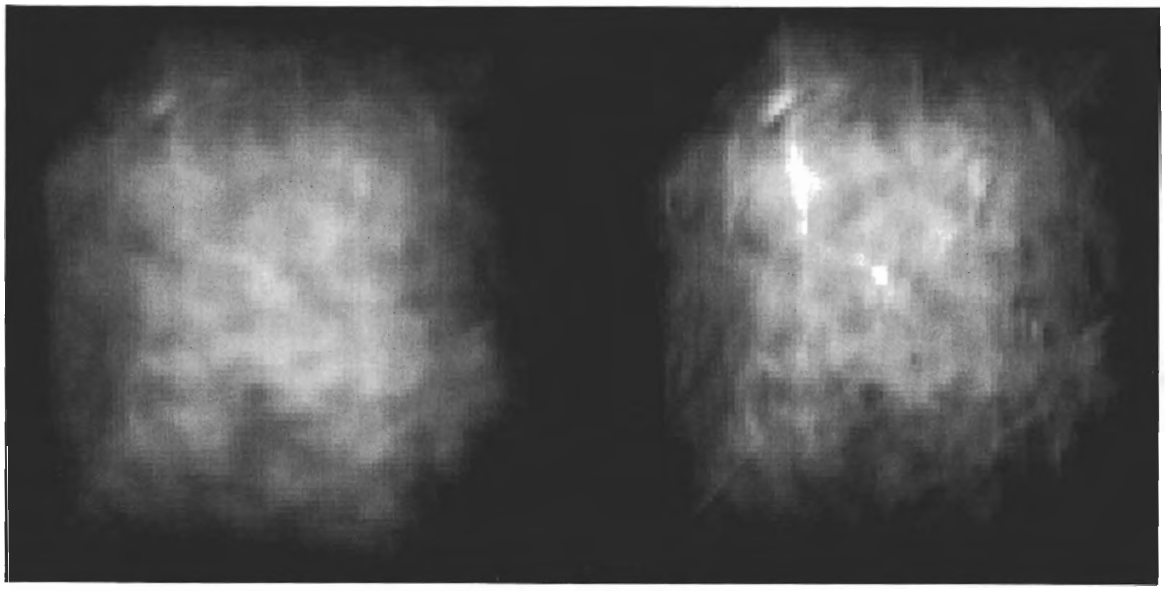

Figure A.4: FVR With and Without Linear Shading. The left image shows normal FVR output. The right image shows the same data and the same view direction with linear shading added.

Totsuka [25] provided a linear approximation to the diffuse lighting coefficient. In our experiments this linear approximation provided sufficient realism for the diffuse lighting effect in fog simulation. The linear approximation used is described by Equation A.2.

$$\max(0, \nabla d(\vec{x}) \cdot \vec{L}) \approx \frac{1}{2}(1 + \frac{\nabla d(\vec{x})}{||\nabla d(\vec{x})||} \cdot \vec{L}) \tag{A.2}$$

We can transform this approximation to the frequency domain using:

$$\mathcal{F}\{\nabla d(\vec{x}) \cdot \vec{L}\} = D(\vec{\omega})\vec{\omega} \cdot \vec{L}$$

Figure A.4 shows two FVR images of a 64^3 fog volume. The left image is rendered without any effects. In the right image, linear shading is used with the light source above and to the left of the viewer.

Creating shadows is another effect that can be achieved with FVR. By rendering an X-ray projection of the fog or smoke as seen from a particular direction, we obtain an attenuation map to use when illuminating the surface for that direction. The result is not a physically accurate shadow because it does not account for scattering effects or multiple light sources, but for applications such as video games and movies it may be sufficiently convincing.

Appendix B

Boundary Conditions for Fluid Flows

This appendix discusses theoretical and numerical treatments of boundary conditions in fluid flows. This is supplementary material to support future work.

B.1 Theory

Boundary conditions describe how a fluid behaves at its interfaces with other substances. These other substances may be either solid objects or fluids with different properties. They may be stationary, in motion caused by the fluid, in motion under their own power, or some combination. In this discussion we will consider only the case of the interface between a fluid and a solid.

At such a boundary the velocity of the fluid is zero, or rather equal to the velocity of the solid; some of the fluid "sticks" to the surface of the solid. This is called the *no-slip condition* and is true for all real fluids provided our scale of interest is large enough for molecular-scale features to be ignored. The no-slip condition is theoretically not true for inviscid fluids or frictionless surfaces, but such things are not found in nature.

A similar condition that applies to both viscous and inviscid fluids is the *no-flow-through condition*, which states that the fluid velocity along the surface normal direction is zero. More intuitively, fluid does not enter or leave the surface. Obviously we might want to violate this condition to approximate a porous material such as sponge, but in reality a porous material is not a simple flat-surfaced solid but one with a very complex interior structure.

The no-slip condition and no-flow-through condition are the two main boundary conditions for fluids against a solid surface. There are additional conditions describing the behavior of properties such as temperature at the interface, and the evolution of the shape of interfaces between different

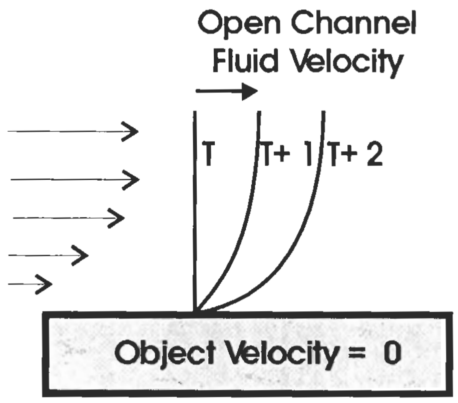

Figure B.1: Fluid Velocity Gradient Near Surfaces. Fluid velocity is equal to object velocity at the surface. Away from the surface it approaches the unrestricted fluid velocity at a rate determined by viscosity. The curved lines represent distortion of fluid volumes by this velocity gradient.

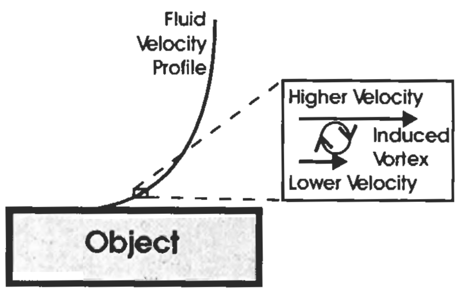

Figure B.2: Conceptual Illustration of Vortex Formation. In regions of high velocity gradient, velocity differences above the fluid's tolerance for shear stress cause volumes of fluid to start rotating, creating vortices.

fluids. The interested reader can consult references such as [15] and [27] for an introduction.

Assuming the bulk of the fluid is moving at a different velocity than an immersed solid, the rate of change of fluid velocity with increasing distance from the solid depends on the viscosity of the fluid. A straight line drawn through the fluid normal to the solid surface at time t will take on a characteristic shape as shown in Figure B.1 at time $t+1$. This shape is called a *velocity profile*. The presence of this velocity gradient near boundaries exerts force both on the solid and on the fluid.

The force exerted on the solid is called *drag* and is an acceleration in the direction of the local fluid flow. Drag strength and direction can vary over the surface of an object depending on its shape and local variations in the fluid velocity. If it is a rigid body and free to move, it will gain linear velocity from the average drag over its surface, and angular velocity as a function of the distribution of drag. It the body is not rigid, it may change shape in response to the drag distribution.

The force exerted on the fluid near a boundary is shear stress, and causes the fluid to deform. All real fluids resist deformation to some degree, and this causes a feedback of force to the object on the other side of the boundary. The fluid's viscosity, or resistance to deformation, determines the strength of the drag it exerts on the solid.

If the viscosity is low, then the velocity gradient near the boundary will be large. Large velocity differences give rise to spinning volumes of fluid, or vortices. To see how this happens, consider Figure B.2. Choose a small volume in a region with a high velocity gradient. The velocity on one side of that volume will be higher than on the other. Particles placed in the fluid on opposite sides will move relative to each other. The relative motion causes shearing of the intervening volume. The fluid will resist the shear force as dictated by its viscosity, and that will partially convert the shearing into rotation. The result is a vortex. So long as the shear force is sufficient to power them and remains in the same direction, vortices persist. Nearby vortices rotating in the same direction will tend to combine to form larger vortices.

The presence of vortices is called turbulence, and it significantly affects the drag on the solid. The amount of turbulence is determined by the fluid viscosity and the roughness of the surface; an irregular surface like a stone wall will give rise to much more turbulence than a smooth surface like plastic. The shape of the surface is also a factor. Sharp edges and regions that induce divergence in the flow are natural forming areas for vortices. Significant effort is invested in designing vehicle hulls that shape turbulence in order to reduce drag and thus save energy or increase speed.

B.2 In Practice

There are several treatments of boundary conditions in numerical fluid simulation. This section will survey a few of them. We concentrate on Eulerian and semi-Lagrangian techniques because they are best suited to graphics applications.

However, there is a fundamental mismatch between Eulerian or semi-Lagrangian fluid simulations and the desire to have them interact with solid objects. Typically objects will be given either as surface representations or in volumetric, sampled form, and will be driven by a Lagrangian physical simulation. It is not clear what to do with fluid simulation cells that are partially intersected by the surface, or that are surrounded on multiple sides by the surface. If the voxelized representation of the surface has anti-aliased regions, should the fluid be allowed to penetrate them? If we simplify the voxelized object by making it a binary discretization, then it is a less accurate approximation of a real object. Is that difference important? We can't expect the objects to move in convenient increments of the grid sample spacing, so how do we update the fluid simulation when the object moves by a large or small amount?

A first approximation of fluid boundary conditions is to ensure that fluid does not enter or leave the surface of a solid object (the no-flow-through condition), that the object's motion stirs the fluid appropriately, and if the simulation includes physical properties such as temperature and density, that the object affects them appropriately.

For a coarse simulation, we may not want to fully enforce the no-slip condition. The layer in which the fluid velocity is equal to the surface velocity is vanishingly thin. Instead of setting the fluid velocity to zero in cells next to a surface, we want to approximate the thin boundary layer by reducing the velocity according to some function of the surface roughness, fluid viscosity and simulation grid sample spacing.

We also want to ensure that the velocity in cells adjacent to the surface is parallel to the surface in order to enforce the no-flow-through condition. Mass conservation for an incompressible fluid states that the amount of matter flowing into any given region is always equal to the amount flowing out. Therefore, the correct thing to do is to set all velocities inside a solid object equal to the object velocity and force all velocities adjacent to the surface to be parallel to the surface, thus making the solid object a region with zero inflow and outflow.

It is not immediately obvious what to do with fluid cells that are bounded on multiple sides by solids. While flow is certainly possible in such volumes, the discrete nature of the numerical simulation combined with the desire to avoid flow into or out of the surrounding solid regions restricts the range of possible solutions. Specifically, the presence of solid faces on the simulation cell restricts the range of flow directions that will not violate the no-flow-through condition.

Foster and Metaxas [5] specify flow velocity on the faces of the simulation grid cells, and incorporate temperature effects. They voxelize their objects and enforce conditions on the voxels along the boundaries of the objects, both inside (solid) and outside (gaseous). For example, a solid heater with a rough surface has zero velocity inside the boundary, zero velocity outside at the surface (causing turbulence), and a high temperature inside which heats the surrounding air as a result of diffusion. Their method requires a binary discretization of the interacting objects, and that voxels adjacent to the objects' surfaces be flagged for special processing.

Using voxelizations of objects to compute their interaction with a fluid is troublesome because re-voxelization is necessary for changes in the shape of the object. Shape changes are common in some animation applications such as video games.

The next major improvement to a flow simulation is to calculate the forces exerted on the object by the fluid, and feed the results back to whatever physical simulation controls the object's motion. This feature is valuable for video game applications, where physical simulation is becoming increasingly important.

Génevaux et al [7] show how to couple a fluid simulation with a physical simulation of rigid and deformable bodies. They use a set of particles transported by the fluid as an interface between the Eulerian fluid simulation and the Lagrangian solid body simulation. They achieve very realistic results demonstrating two-way energy exchange between the fluid and solids, but achieving such realism requires hand-tuning several interrelated parameters.

The boundaries of the simulation grid are often treated as wrap-around (periodic) in the graphics-oriented literature, especially in Stam's work [17, 18, 19]. This is convenient for frequency domain techniques and can be useful for generating images that tile seamlessly, but it is not physically realistic and the tiling effect can cause undesirable visual artifacts in many applications. Fortunately a periodic simulation can easily be converted to a non-periodic one simply by adding solid barriers around the edges.

Objects moving through space add velocity to the fluid surrounding them. This much is easy to do in the frequency domain; translation of a voxelized object is equivalent to phase-shifting its frequency representation, and addition is the same in both domains. Rotational motion is expressed by rotating the frequency domain representation of an object.

However, for shape changes the frequency domain picture is less clear. It may be necessary to re-voxelize objects when this happens, and compute the new forces in the spatial domain. It is also not clear how to enforce the no-flow-through condition in the frequency domain. These questions are left to future investigators.

Bibliography

[1] R. N. Bracewell. *Two Dimensional Imaging.* Prentice Hall Inc., 1995.

[2] A. Entezari, R. Scoggins, T. Möller, and R. Machiraju. Shading for Fourier volume rendering. *Proceedings of Symposium of Volume Visualization and Graphics*, pages 131–138, October 2002.

[3] C. Evangelinos and G. E. Karniadakis. Communication performance models in *prism*: A spectral element–Fourier parallel Navier–Stokes solver. *Supercomputing*, November 17–22 1996.

[4] R. Fedkiw, J. Stam, and H.W. Jensen. Visual simulation of smoke. *SIGGRAPH*, pages 23–30, 2001.

[5] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. *Computer Graphics*, 31(Annual Conference Series):181–188, 1997.

[6] M. Frigo and S.G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.

[7] O. Génevaux, A. Habibi, and J.M. Dischler. Simulating fluid–solid interaction. *Proceedings of Graphics Interface*, June 2003.

[8] P. M. Gerhart and R. J. Gross. *Fundamentals of Fluid Dynamics.* Addison–Wesley, 1985.

[9] F. Koster, K. Schneider, M. Griebel, and M. Farge. Adaptive wavelet methods for the Navier–Stokes equations. *http://wwwwissrech.iam.uni-bonn.de/research/pub/koster/berlin99.ps.gz*, 1999. (Last access August 2003).

[10] S. Lapierre, A. Entezari, and T. Möller. Practicalities of Fourier-domain fluid simulation. *http://www.cs.sfu.ca/~slapierr/personal/documents/fdpaper.pdf*, 2003. (Last access August 2003).

[11] T. Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[12] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical comparison of popular volume rendering algorithms. *Proceedings of the 2000 Symposium on Volume Visualization*, pages 81–90, October 2000.

[13] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *SIGGRAPH, ACM Transactions on Graphics*, pages 721–728, 2002.

[14] R. Peyret and T. D. Taylor. *Computational Methods for Fluid Flow.* Springer–Verlag, 1983.

[15] S. M. Richardson. *Fluid Mechanics.* Hemisphere, 1989.

[16] H. Ritchie. Eliminating the interpolation associated with the semi–langrangian scheme. *Monthly Weather Review*, 114:135–146, 1986.

[17] J. Stam. Stable fluids. *SIGGRAPH Annual Conference Proceedings*, pages 121–128, 1999.

[18] J. Stam. A simple fluid solver based on the FFT. *Journal of Graphics Tools*, 6(2):43–52, 2001.

[19] J. Stam. Real–time fluid dynamics for games. *Proceedings of the Game Developers' Conference*, March 2003.

[20] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. *SIGGRAPH Annual Conference Proceedings*, pages 369–373, 1993.

[21] A. Staniforth and J. Cõté. Semi–Lagrangian integration schemes for atmospheric models – a review. *Monthly Weather Review*, 119(9):2206–2223, September 1991.

[22] P. Stark. Fourier volume rendering of irregular data sets. MSc thesis, Simon Fraser University, 2001.

[23] J. Steinhoff and D. Underhill. Modification of the Euler equations for "vorticity confinement"; application to the computation of interacting vortex rings. *Physics of Fluids*, 6(8):2738–2744, 1994.

[24] J. Strain. Semi–Lagrangian methods for level set equations. *Journal of Computational Physics*, August 1998.

[25] T. Totsuka and M. Levoy. Frequency domain volume rendering. *Computer Graphics*, 27(4):271–278, August 1993.

[26] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics*, 22(3), July 2003.

[27] R. J. Le Veque. *Numerical Methods for Conservation Laws.* Birkhauser Verlag, 2 edition, 1992.

[28] P. Witting. Computational fluid dynamics in a traditional animation environment. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 129–136. ACM Press/Addison-Wesley Publishing Co., 1999.

[29] R. Yagel, D. M. Reed, A. Law, P. Shih, and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Proceedings 1996 Symposium on Volume Visualization*, pages 55–62, 1996.

Index