Vortex Based Smoke Simulation and Control

by

Derek Nowrouzezahrai

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Vortex Based Smoke Simulation and Control

Derek Nowrouzezahrai

Master of Science

Graduate Department of Computer Science

University of Toronto

2006

The simulation and control of natural phenomena is an important area of computer graphics that has applications in animation and interactive entertainment. We present a novel method of replicating the physical behaviour of wispy smoke using a computational simulation of the evolution of a vorticity field. We represent this field compactly as multi-resolution closed curves we call filaments. We extend the applicability of our technique with new artistic control mechanisms that operate on filaments. Using these controls, an animator can manipulate the underlying flow fields. Unlike previous techniques, our control framework allows an artist to visualize the effects of different control layouts interactively. We also present a real-time rendering algorithm and shading model for the particles used to represent the smoke in an animation.

Dedication

To my parents, Mary and Mohsen.

Acknowledgements

Writing a thesis is just a matter of typing the right set of keys in the right sequential order; unfortunately, it seems no matter where I was, I was surrounded by dysfunctional keyboards.

Now that it's done, I would like to thank those who helped me along the way. My supervisor, Eugene Fiume, is an energetic and motivational person and I would not have been able to finish without his support. He was always available to answer my questions, point me in directions I would never have thought about, and (most importantly) to chit chat about things other than my thesis.

I'd like to thank Sonya for her patience and for listening to my ideas and my whining. Her company played a key role in helping me keep my mind on task through much of the writing and research put into this thesis.

Luckily I was able to meet Alexis Angelidis during my first semester of graduate studies and he guided me through much of the background knowledge required in fluids and shape deformations. He graciously invited me to work with him on the project that would eventually build into this thesis. I would not have produced this work had he not been here. Alexis originally worked on the idea of using vorticity-carrying filaments as an alternative method of simulating the behaviour of wispy smoke, along with Fabrice Neyret. Alexis introduced me to the realm of animation control for physical simulations and gave me an insider's view on the details of his ideas and, more importantly, the methods by which he arrived at the ideas. I am reminded of Isaac Newton's famous words: *If I have seen a little further it is by standing on the shoulders of giants.* Maybe with time I will see a little further, and if so, it will be thanks to Alexis' help early on in my research career.

Although I've been interested and involved in computer graphics since I was young, the first Professor to show me that computer graphics is (much) more than about making "pretty pictures" was Gladimir Baranoski. His continued support, insight and sense of

humour remain with me throughout all my work in this area. He instilled the importance of scientific validation and proper research methodologies early on in my research career.

The Dynamic Graphics Project is filled with interesting and fun people, each of which helped me maintain my sanity. Patricio, Noah, Jacky, the Mikes, Ian, Alex, Christian, Anand, the Dans, Pierre[1], Tristan, Nigel, Shazad and Jack to name a few. Thanks for the laughs, beers and for just plain putting up with me[2].

I am also lucky to have worked with two very distinguished Professors on this project, Karan Singh and Fabrice Neyret.

[1]Thanks for the free housing that I needed for the last month of writing.
[2]This is harder than it sounds.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The diversity of research areas in computer graphics is a testament to the applicability of computer graphics and computer science theory to interesting problems. The computer graphics field has evolved into a cross-disciplinary area with contributions to and from engineering, mathematics, computer science, machine learning, physics and chemistry to name a few. These cross-contributions have at times blurred the line between the different fields. One such marriage of research areas that has grown into a full-fletched sub-area of computer graphics is between computer animation and computational fluid dynamics. This thesis addresses a problem space within this frame.

Computer animation is a term broadly used to describe any dynamic interaction of two or three dimensional phenomena with each other and/or their environment. Computer animation tools and research are employed on a wide array of real-world applications such as video games, live-action and animated films, medical and physical simulations and education. Both general and specialized animation tools exist for designing and breathing life into a myriad of creative outlets.

The field of computational fluid dynamics investigates the underlying mathematical models driving the physical interaction of gases and liquids with each other and solids. This field focuses on providing the mathematical framework of fluid flow; visualization is

of minor interest, leaving a gap that computer graphics is especially well suited for.

Initially, applications of computer graphics in fluid dynamics focused on using simplified mathematical models of fluid behaviour and coupling the results of these simulations with graphical output. Research in this area of computer graphics progressed from simple two-dimensional models to three-dimensional models, with emphasis slowly moving towards both more faithful mathematical representations and enhanced acceleration techniques. Issues such as numerical stability, alternative storage methods for acceleration and visualization techniques became the main research focuses. At this point, the graphics CFD community has a firm grasp on the basics of multiple-interface simulation as well as rendering techniques for different media. Initializing a fluid-dynamical system and yielding a realistic final rendered animation is now commonplace.

The next logical progression of the area was to devise methods for *controlling* these simulations to increase their flexibility and their potential for creative expression. This thesis focuses on work in this area. The mathematical foundation of fluid dynamics in computer graphics facilitated the application of several different methods for adding artistic control to fluid simulations. As we shall see, several different approaches to constraining the behaviour of fluids have been investigated. Ideally, the success of any particular method should be measured by its efficacy, the quality of its results, and the degree to which it is taken up by artists or other users.

1.1 Statement of Thesis

This thesis describes a novel method for controlling the behaviour of smoke simulations with particular focus on producing compelling animations of *wispy* smoke effects. The smoke animation system presented in this thesis achieves interactive modeling performance, unlike most mainstream smoke simulation/animation packages currently used. Furthermore, we introduce a unique multi-level flow field manipulation architecture that

allows an artist to translate animation content generated from a low-frequency flow field into a higher-frequency flow field animation with little or no additional work, unlike most existing simulators which confine a user to their initial system resolution.

The majority of production quality fluid simulation and animation tools are based on an *Eulerian* formalism (see chapter 3) that discretizes space in order to determine the flow of a fluid in a system. Our system is based on an alternative method, the *Lagrangian* formalism, that traces the physical behaviour of the system at markers in space carried by the flow. Although our system is capable of generating compelling animations of thick, billowing smoke, we instead focus on the design of animations involving *wispy* smoke. An example of wispy smoke would be the smoke from a cigarette or an incense stick. Figure 1.1 below contains real photographs of wispy smoke.

Figure 1.1: Images of real smoke, courtesy of William Brennan [10].

We will introduce a multi-resolution basis used to represent filaments that carry the underlying flow field of our physical system. Moreover, our basis is compact and is user-controllable. We outline different control mechanisms that are coupled to our basis, allowing for intuitive manipulation of the flow field while maintaining the physical plausibility of the resulting animation sequence. We introduce control tools, such as current control curves and current attractors, to suit specific requirements of artist while also enabling them with interfaces that are easy to use and similar to existing computer modelling and animation tools. A novel shading algorithm leverages the benefits of a smoke

particle representation. A high level of realism can be achieved with accelerated graphics hardware that allows the interleaving of designing and observing an animation, which results in what we hope will be a more productive and enjoyable animation workflow for the users of our system.

Although a great deal of this thesis will focus on the mathematics and engineering aspects behind our animation system, usability is still one of the driving forces behind our motivation.

1.2 Contributions

The content presented in this thesis is an extension of the work of Angelidis *et al.*, "A Fast, Stable and Controllable Basis for Smoke Animation" [5]. Since the completion of that research, several insights and improvements over the publication were undertaken. These changes are reported in this thesis. Some equations and figures are used (and cited) verbosely from the work of Angelidis *et al.*.

We describe a smoke simulation and animation system with the ability to dynamically scale visual and physical complexity. We provide an interactive design workflow that will enable artists to generate complex smoke animation sequences in a fraction of the time currently required by production fluid animation packages. We briefly introduce the three fundamental contributions of our work below.

1.2.1 Multi-resolution Basis

We chose a Lagrangian formalism (see chapter 3) to describe the flow of our physical system. Specifically, following research in the area of Computational Fluid Dynamics, we work from the abstraction that the flow of physical smoke systems can be represented with a small set of curve-like filaments.

We present a novel and compact multi-resolution basis for representing each filament.

Our basis not only concisely represents the underlying flow field of the system, but also couples to some effective control mechanisms that we shall introduce.

1.2.2 Control Mechanisms and Tools

The Lagrangian method of representing the flow behaviour, coupled with the efficient advection algorithm we use with our compact multi-resolution basis provides a fast simulation engine for the evolution of various smoke densities. However, our goal supersedes mere simulation and incorporates animation.

We introduce two control mechanisms, *turning* and *paddling* (see chapter 4), that operate on individual filaments (represented in our basis) producing the required flow modification. It is important to note that these mechanisms aim to balance the contradicting requirements of artistic influence and physical constraints on the smoke flow. For example, we want to be able to move smoke in certain directions, however not at the expense of disturbing the perceived realism of the effect.

The low-level mechanisms do not provide an intuitive form of control over the smoke and an artist can find them very difficult to use. Unlike many smoke animation techniques, we abstract the low-level control mechanisms with higher-level control *tools* that are both intuitive to use and familiar to artists.

Furthermore, thanks in part to the ability of scaling the complexity of our simulation in real-time, an artist can apply controls to the smoke filaments and manipulate these controls interactively as the simulation runs. This leads to a design workflow that provides feedback to the artist interactively, instead of the currently accepted *edit-wait-view* workflow.

1.2.3 Acceleration

Acceleration methods for fluid simulations in computer graphics is a open research area, although many substantial results have been presented. We employ a set of acceleration

techniques, some implemented in software and some implemented as parallel execution units on commonity graphics hardware architectures.

Using graphics hardware to accelerate physical simulations for computer graphics applications is slowly becoming an accepted area of work and we leverage the increasing availability of both graphics hardware resources and their associated development tools.

1.3 Thesis Organization

We have segmented our literature review into two groups. Chapter 2 gives an overview of some key works in the area of fluid dynamics for computer graphics. The work described in that chapter is by no means a complete taxonomy. Instead, we make occasional reference to these works in future chapters. Previous work in specific areas of particular relevance is presented as needed at the beginning of chapters three, four and five in order to guide the flow of the topics to the reader.

Chapter 3 focuses on the basics of computational fluid dynamics in computer graphics, as well as the two main formalisms used. We also discuss the specific concerns of our choice of representation: Lagrangian filaments.

Our multi-resolution basis for describing filaments, as well as the control mechanisms and tools we apply to control filaments are described in detail in Chapter 4.

Chapter 5 is reserved for the details of our novel adaptive smoke particle representation as well as the efficient shading algorithm designed specifically for our particle representation.

Chapters 6 and 7 briefly overview the software engineering aspects of our implementation, including acceleration techniques, and some results generated by our system.

Lastly, we conclude our work in Chapter 8 and discuss diverse possibilities for future work.

Chapter 2

Survey of Significant Contributions

We divide the summary of significant contributions in fields related to computational fluid dynamics in computer graphics. This chapter will focus on some founding work in the areas of simulation and animation of fluids in computer graphics. Some key research is deferred to later discussions in Chapters 3, 4 and 5. Summaries of these works are stressed in future chapters since they tie into or provide special background for the topics discussed in those chapters.

This chapter is divided into two sections. The first focuses on some founding work in the area of fluid simulation in computer graphics. The second section covers the seminal works in fluid animation in computer graphics.

2.1 Seminal Works of Computational Fluid Dynamics in Computer Graphics

One trend that is observed in most areas of computer graphics is the increasing complexity of research models. For example, in rendering, very flexible yet ad-hoc physically-motivated reflection models were slowly replaced by physically-based and data-driven reflection models. A similar trend can be observed in our field, in that early ad hoc

models have been supplanted by more rigorous models. This evolution will be visible in the presentation of the works in this chapter and this section.

In 1986, Fournier and Reeves detailed an ocean wave model based on the Gerstner [28] or Rankine [64] model of elliptical stationary orbits [25]. "A simple model of ocean waves" is a parameterized time-dependent model that generates parametric surfaces resembling ocean waves. These surfaces react to different ocean floor arrangements. Previous attempts of modeling ocean waves used Gaussian distributions and height fields. However, breaking waves are an example of a wave that cannot be represented with a simple height field. Fournier and Reeves use a Lagrangian particle tracking method to represent the ocean wave surfaces and trace their evolution. A tracked particle is assumed to represent a sphere about itself and a parameterized sinusoidal evolution function is used to propagate each particle.

Kass and Miller present a water simulation technique based on the *shallow water equations* [41, 79]. These equations are significantly simpler than the Navier-Stokes equations, but the same spectrum of effects that earlier methods (such as Fournier and Reeves' method) are still reproducible with these equations. Unlike earlier work, the system presented by Kass and Miller is robust enough to handle more complicated effects, such as boundary conditions against objects of variable topology [41]. The system outputs a height-field. This presents a trade off: effects such as breaking waves cannot be simulated but on the other hand the height-field representation affords many computational benefits. The most significant of these benefits is that, after discretization and the application of a first-order approximation of the shallow water differential equations, the evolution of the fluid during each time-step is conducted by simply solving a linear system of equations. The yielded solution is implicitly stable.

Miller and Pearce simulate viscous fluids, such as lava or clay, using a connected particle system in their work "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids" [54]. Each connected particle has a mass, radius, temperature,

current position and velocity associated to it. A particle's position and velocity are affected by inter-particle attraction and repulsion forces defined in the paper. This work is similar to the work of Smoothed Particle Hydrodynamics (see chapter 3) with distance fall-off kernels modified to produce different types of simulated viscous materials. Miller and Pearce also handle simple object interaction by enforcing collision constraints on each particle with each object in the system.

In 1992, Shinya and Fournier detail a three component system for delivering a wind field with useful properties [73]. "Stochastic Motion - Motion Under the Influence of Wind" derives from models of wind behaviour used in engineering for ensuring the structural integrity of large buildings and bridges. One of the key contributions of this paper, as it relates to our work, is a thorough frequency analysis of wind fields yielding a compact and meaningful representation of spatio-temporal variations in the wind field (see section 4.4). The work applies their frequency representation of wind with applications to structural models (such as grammar-based representations of plants or trees), particle systems, two dimensional artistic texture models and three dimensional textures.

In chapter 4 we summarize an important contribution by Stam and Fiume outlining a dual-resolution analysis of gaseous turbulent behaviour. Just before the work of Stam and Fiume, Sakas presented a physically-motivated model based loosely on a flavour of work similar to the noise texture synthesis work of Perlin in 1985 [70, 59]. "Modeling and Animating Turbulent Gaseous Phenomena Using Spectral Synthesis" synthesizes fluid-like fields in frequency space and also provides an interesting intuition linking the components of a discrete frequency spectrum with spatial eddies in the flow field. This intuitive concept is applied to a simple field constructed using overlapping sinusoids in frequency space, all while satisfying some basic mathematical constraints [70].

Two of the most referenced works in fluid simulation in computer graphics are both authored by Foster and Metaxas in 1996 and 1997. The first work, "Realistic Animation of Fluids" [23], introduced a two and three dimensional *liquid* simulation system based on

the Eulerian formalism. Thus, uniform two and three dimensional grids are used to store and update the density and velocity values of the system. The Navier-Stokes governing equations of fluid dynamics are used as a mathematical basis of this work.

Foster and Metaxas also introduce the embedding of rigid obstacles in the flow by assuming that obstacle boundaries are aligned with grid cell boundaries. As summarized in chapter 3, Foster and Metaxas use a finite differencing scheme to solve for the free variables of the Navier-Stokes equation and update the values on the grid. Furthermore, massless surface tracking particles (in two dimensions) and heightfields (in three dimensions) are used to track the fluid's surface. A time-dependent pressure field can be used to introduce external forces in the simulation. Foster and Metaxas use a convergence criterion and an iterative algorithm to ensure that the stability of the system is maintained (on a local scale). Although the system presented in this paper paved the way for realistic fluid simulations coupled with equally faithful computer graphics renderings, the convergence algorithm and dependence on grid cell size became obvious limitations. Chapter 3 will summarize the "Stable Fluids" work of Jos Stam [74] that introduced a simulation technique that ensured unconditional stability, regardless of the choice of time step size, without requiring a numerical convergence algorithm.

Following the success of their first work, Foster and Metaxas presented "Modeling the Motion of a Hot, Turbulent Gas" [24] in 1997. This work focused on the interaction of hot gases with their surrounding (rigid) environment, and thus introduced an extra layer of complexity into the simulation procedure since explicit treatment of the effects of variable temperature had to be taken into account. Foster and Metaxas emphasize the physical foundation used to generate effects due to thermal buoyancy, convection and turbulence with a greater level of realism than results that were previously left up to manual artistic design.

The mathematical model introduced by Foster and Metaxas combines (a reduced form of) the Navier-Stokes governing equation of fluid dynamics and an equation for the

differential-temperature turbulence effects. The Eulerian grid-based approach Foster and Metaxas use is similar to other standard Eulerian gas simulators, with the addition of a local (per cell boundary) external force that estimates the effects due to differential temperatures within the gas. This force is modeled as a parameterized weighted difference between adjacent cell temperatures. The temperature of the gas is tracked in the grid using a second set of differential equations. A finite-differencing approach is used to approximately solve these equations and evolve the temperature of the gas in the grid [24].

Foster and Metaxas presented a study in the error introduced due to grid resolution and also enforce stability by choosing sufficiently small time steps. The elaboration on the method of determining the size of the time step required to maintain stability would prove to be a motivating discussion for future work that would aim to eliminate any constraints on time step size in hopes of providing an arbitrarily stable simulation. Notably the work of Jos Stam (see chapter 3) addressed this issue and paved the road for the next generation of fluid simulations in computer graphics.

Recently, researchers from the University of California at Berkeley presented multiple works on the simulation and animation of fluids using tetrahedral mesh simulators. The first such work by Feldman *et al.*, "Animating Gases with Hybrid Meshes", introduced enhancements to standard Eulerian grid simulators allowing for tetrahedral and hexahedral cell arrangements to be simulated efficiently [20]. This allows the fluid to not only be simulated within arbitrary volumes, but obstacles of arbitrary construction can also interact with the fluid without requiring adaptive or fine-scale discretization near fluid-object boundaries. The main contribution of this work is the mathematical extension presented for solving the Navier-Stokes equations across both tetrahedral and hexahedral cells. Furthermore, coupling of these cells with standard cube cells is also presented. A high-order moving least squares interpolation technique is also introduced for both new types of discretization. A second work is concerned with the animation and control of

simulations on such hybrid meshes, which we further discuss in section 2.2.

Unlike Feldman *et al.*, the work of Elcott *et al.* in 2005 simulates fluid flow within tetrahedral meshes using discrete geometric operators and does not require an explicit velocity reprojection step to maintain the divergence free property of the velocity field [69]. In fact, all Eulerian fluid simulators in computer graphics use a fractional step method of solving the Navier-Stokes equations; this technique requires the explicit reprojection of advected velocity fields onto divergence-free fields [69]. Ignoring the additional computational cost of this reprojection, drawbacks also include the introduction of numerical dissipation into the flow. Vorticity injection, vorticity confinement and adaptive discretization techniques have been introduced in computer graphics to try to minimize this energy loss (see chapter 3 for a summary of these works), but these additional computations only add to the cost of simulation. On a high level, the technique of Elcott *et al.* is similar to our technique in the sense that both avoid the explicit reprojection step. Despite the similarities between our technique and that of Elcott *et al.*, our technique is generally different as we will elaborate in later chapters.

Elcott *et al.* introduce a novel geometric solution to the simulation of fluid flow that handles complex object boundaries and fluid simulation domains at fairly low computational cost. The vorticity formulation of the Navier-Stokes equations (see chapter 3 for mathematical details of this formulation) are solved in an Eulerian fashion along meshes of arbitrary topology and representation (grid, tetrahedral or hexahedral for example). The method of Elcott *et al.* preserves the *circulation* of a fluid (see chapter 3) and the discrete operators have a small region of support, resulting in the generation of sparse linear systems that can be solved quickly. Circulation preservation also guarantees that no numerical dissipation occurs; the system presented is also stable for arbitrarily long time steps.

2.1.1 Rendering

Rendering fluids and gases presented an interesting problem to computer graphics researchers 25 years ago who, at the time, were only typically exposed to visualizing solids. Furthermore, the clear cut geometrical representations of objects left no room for interpretation when it came to what a shape should look like.

The work of Fournier and Reeves mentioned above, despite being 20 years old, uses many of the advanced rendering techniques that are currently in use today. A Fresnel reflection model is used with environment map lighting to obtain an accurate direct lighting result. To reduce the sequential nature of the final renderings, Fournier and Reeves also perturb the normals of the generated surface with a bump map; the bump map is advected in a similar fashion as the waves themselves, to maintain a certain consistency between the coherence of the waves [25].

Kass and Miller use a *caustic shading* method to approximate the appearance of water in their work. Their method assumes that the underlying terrain covered by the fluid is locally flat and the so-called *flat-bottom approximation* is used to approximate the intersection of incoming light rays at the interface between water and air [41]. With these two simplifications, an illumination map is generated on the terrain and combined with a convolution kernel operation to generate the final rendering. This results is a highly approximated result at the benefit of extremely fast rendering times.

Miller and Pearce choose to determine a surface for visualizing their connected particles by simply defining a sphere with a radius defined by a potential function at each particle, as opposed to determining the actual isosurface of the linear combination of all the field functions. Today's modern graphics hardware acceleration is capable of rendering iso-surfaces of blobbies in real-time [13].

Building on the foundation laid by his work on the rendering equation [39], Kajiya presented a mathematical model based on radiative scattering theory with Von Herzen in 1984 for accurately ray tracing general volume densities [40]. Their mathematical

exposition laid the foundation for future work in accurately rendering participating media and also presented results of unparalleled quality for its time.

In 1990, Ebert and Parent demonstrated a system for combining volumetric and scan-line rendered objects using the accumulation buffer and accelerated shadow table generation for environmental and self shadowing of gases on scanline converted objects [17]. Of note, Foster and Metaxas used massless marker particles to visualize the flow of the turbulent, temperature-dependent motion fields generated by their simulator in [24] and, after determining a volumetric density map from the particle distribution, use the method of Ebert and Parent to render the gas in their system.

2.2 Work in Controlling Fluid Simulations

A significant contribution of this thesis is to propose a set of underlying mathematical mechanisms and intuitive artist-controllable tools built using these mechanisms to control the behaviour of a smoke simulation. Chapter 3 summarizes some key work in the area of fluid control in computer graphics and animation. Below we outline some works not covered in chapter 3. These works lay the foundation on which the future works in fluid animation and control were built.

Ebert *et al.* extend the work of Ebert and Parent [17] and present fluid animation tools based primarily on tabulated flow manipulation volumes. Of specific interest in this work is the introduction of the notion of an attractor tool for fluid control. Ebert *et al.* use attractors defined with centers and radii of influence to gradually draw a fluid or gas towards a desired location. One of the smoke control tools we present in this thesis is conceptually similar to Ebert *et al.*'s attractors. The simulation engine underlying the control mechanisms outlined in this work is, like the work of Sakas discussed earlier, based on three-dimensional spectral noise synthesis techniques and is physically-motivated [17].

In 1991, Wejchert and Haumann presented a simple model of air turbulence based

on simplifications of the Navier-Stokes equations for irrotational, inviscid and incompressible fluids [82]. "Animation Aerodynamics" takes advantage of the linear nature of the resulting Laplace equation required to advect the fluid's velocity field to accelerate boundary condition calculations. Simple object models, such as spherical particles, reduce from the total computational cost and allow this simple model to generate results quickly. Wejchert and Haumann define four *flow primitives* which may be arbitrarily combined to yield final flow behaviour. This is a simple, yet effective, form of control. Wejchert and Haumann explicitly detail their speed versus accuracy trade-off; the irrotational assumption they make on the flow behaviour of outdoor air currents precludes the representation of turbulent behaviour, which in time has become a standard requirement of fluid simulations.

Depicting the effect of fire also became an interesting problem for computer graphics researchers in the early 1990's. Chiba *et al.* were the first group to bring this problem into the spotlight [11], however the work of Lamorlette and Foster is perhaps the most influential papers in this area of fluid simulation in computer graphics. "Structural Modeling of Flames for a Production Environment" [46] describes a robust production-level multi-stage flame simulator. The system is a combination of a physics-based simulator and data-driven simulator with multiple noise sub-simulations added at separate stages of the simulation. Flames are represented by parametric curves which are re-sampled to maintain continuity. These curves are the elementary objects evolved in time by the simulator. Moreover, the curves can sub-divide and spawn child curves based on a heuristic approach. An iso-surface of the flame is obtained from particles obtained using a density fall-off kernel applied to the curves. Lamorlette and Foster apply procedural noise at the particle-level. A second level of turbulent detail is added to the particles after a transformation stage. Artistic control of the flame is delivered via user-definable wind fields and pre-defined procedural control tools.

Pighin *et al.* presented a novel spatio-temporal reparameterization technique for

fluid simulations that allowed for post-simulation editing of fluid animation sequences in "Modeling and Editing Flows using Advected Radial Basis Functions" [60]. The results of an Eulerian simulation are post-processed and a set of radial basis functions are fit to the variation of density and temperature of particles advected through the flow. Once fit, an artist can directly manipulate the location and bandwidth of the radial basis functions and a secondary Smoothed Particle Hydrodynamics simulation is run to ensure the appropriate physical constraints are maintained. Moreover, the traced particle trajectory paths may also be manipulated by an artist.

Recently, Rasmussen *et al.* introduced two extensions to a standard particle level-set fluid simulator targeted at delivering the artistic control required to reproduce *sloppy* liquids [65]. In "Directable Photorealistic Liquids", Rasmussen *et al.* define a sloppy liquid as one with both hard and soft enforced interface constraints. Handling these two types of constraints is the main contributions of this work. The work also details changes to a standard particle level-set fluid simulator required to support the additional resulting fluid behaviour, such as merging fluid blobs and proper interaction with thin submerged objects. Control is presented to an artist in the form of spherical and cylindrical control shapes and the ability to control the degree of adherence to these flow directing shapes via velocity and viscosity constraint manipulations. Technically, the paper details how this level of control is integrated into current high-end fluid simulators while preserving the mathematical requirements of the generated velocity fields. The artist can define the surface of a fluid with hard velocity constraints while allowing other portions of the fluid volume to flow more freely with soft directed velocity constraints. Lastly, a novel grid windowing scheme and level-set boundary interpolation methods are introduced.

Shi and Yu introduce a method of directing the shape of a moving fluid according to animated target shapes while preserving the realism of the generated flow [72]. "Taming Liquids for Rapidly Changing Targets" introduces two additional types of forces to the fluid simulation system in order to guide the fluid towards the various input shapes.

Firstly, a *feedback force* is defined across regions within the fluid and on fluid boundaries. This force is modeled as the combination of a simple proportional-derivative controller used to maintain velocity constraints and a shape matching force obtained using an optimization procedure. Both of the velocity control and shape maintenance force fields are calculated such that the resulting field is divergence free. A secondary *potential force* field is defined geometrically about the target shape and its skeleton. Of note, since artistic influence is defined using animated target shapes, it is difficult to design a fluid flow that is completely natural. Instead, this type of system is mostly suitable for generating flows that look believable but are physically improbable.

Building from their advancements of fluid simulations with tetrahedral and hexahedral mesh domains, Feldman *et al.* presented a system for animating fluids using their augmented geometric fluid simulators in "Fluids in Deforming Meshes" [21]. The simulator presented in this work applies a straightforward addition to the work presented in [20], that we described earlier in this chapter. Instead of simulating the fluid flow on static hybrid meshes composed of cubic, tetrehedral and hexahedral cells, the meshes are deformable during the animation sequence. Handling these deformations simply involves remeshing the hybrid meshes and correctly handling the semi-Lagrangian backwards particle tracing[1] along the altered mesh segments. All other simulation issues, including boundary conditions, are handled exactly as in [20] and the authors explain how the computations required to perform the remeshing and correct particle tracing account for less than five percent of the total simulation time (on top of the time of the original method of [20]). This technique allows an artist or geometric modeler to animate a mesh and expect realistic response of a fluid against that mesh. An excellent example of a practical application of this method is provided in the paper; a smoke-filled Buddha statue's stomach is contracted and expanded, causing the illusion of smoke being exhaled out of the Buddha's mouth.

[1]See chapter 3 for an indepth summary of semi-Lagrangian particle tracing, as introduced in [74].

2.3 Summary

We summarized some of the key works in simulating and controlling liquids and solids in computer graphics. Chapters 3 and 4 supplement the previous work presented in this chapter with other key works of note in the field. A few trends, which will be emphasized further on in the course of thesis, emerge in the works presented in this chapter. The majority of the simulation techniques are Eulerian methods that, despite optimization, typically still run in time complexity $\mathcal{O}(V)$, where V is the size of the discrete volumetric domain of simulation (or area for two-dimensional simulations). This complexity often precludes the real-time simulation of complex flow behaviour. Furthermore, none of the techniques presented above allow an artist to manipulate the flow behaviour with real-time feedback. Some of the animation and control works presented make efforts to provide artists with tools that are easy-to-use, but this is often an under-investigated and tertiary contribution.

Our thesis presents a Lagrangian technique for simulating and controlling smoke with scalable interactive artistic control using intuitive tools.

Chapter 3

Methods of Fluid Simulation

In order to *simulate* the complicated behaviour of fluids, we must first familiarize ourselves with the mathematical equations which govern these flows as well as the numerical techniques used to solve these equations. Fortunately, the field of computational fluid dynamics (CFD) is the source of a large body of research in the mathematical representation of different types of fluid flow. As with many areas in computer graphics, an appropriate trade-off between the accuracy and speed of the mathematical procedure used to simulate the fluids must be balanced.

This chapter will introduce the mathematical framework for modeling fluid behaviour as well as elaborating on the two numerical formalisms commonly used to solve these equations. Our method of simulation (see section 4.2.2) is based on one of these formalisms. For the remainder of this thesis, we will assume that the fluids we are aiming to simulate and control are both *incompressible* and *inviscid*.

A fluid is said to be incompressible if volume is consistently preserved on a local and global scale. Incompressibility can be more easily related to the mathematics of fluid flow as a numerical constraint that the density of a fluid remains constant. Interestingly, all fluids act incompressibly, within a tolerance of approximately 5%, when their velocity remains below Mach 0.3 [68]. We will only consider the behaviour of an *ideal fluid* that

is inviscid, and thus have no viscosity, which simplifies the mathematics significantly. Moreover, although the focus of this thesis is on the simulation and control of smoke, both the mathematics and numerical solving methods presented in this chapter can be extended to simulate water and other liquids, although special attention must explicitly be applied to tracking the boundaries between liquids and gases. Boundary tracking is not required in the simulation of smoke.

3.1 Eulerian and Lagrangian Formalisms

The Navier-Stokes equations, sometimes referred to as *field equations*, govern the behaviour of fluid materials with the independent variables being the density, temperature and velocity fields in the fluid's continuum. For incompressible fluid flow, these equations are:

$$\nabla \cdot \vec{u} = 0 \tag{3.1}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{f} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u}, \tag{3.2}$$

where $\vec{u}$ is the velocity vector, $\vec{f}$ is the sum of external forces (such as gravity), ρ is the density, p is the pressure and ν is the kinematic coefficient of viscosity. The ∇ symbol can be used to represent a vector of partial derivatives (spatial derivative in our case): $\nabla = (\frac{\partial}{\partial \vec{x}}, \frac{\partial}{\partial \vec{y}}, \frac{\partial}{\partial \vec{z}})$. Moreover, a similar notation, $\nabla\cdot$, is used to denote the *divergence* of a vector field and $\nabla\times$ denotes the *curl* of a vector field. Note that $\nabla^2 = \nabla \cdot \nabla$. The inviscid form of the Navier-Stokes equation is very similar, except the added constraint of $\nu = 0$ is applied, yielding a modified form of equation 3.2:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{f} - \frac{1}{\rho}\nabla p. \tag{3.3}$$

Equation 3.1 is obtained by applying the law of conservation of mass to the fluid and equation 3.2 is obtained by applying the law of conservation of momentum to the fluid. Deriving these equations is beyond the scope of this thesis, although any standard text on fluid mechanics, such as [68], elaborates on these derivations in detail. Thus, as represented in the Navier-Stokes equations, we see that a fluid's flow can be tracked by monitoring the evolution and coupling of the time-varying velocity and pressure fields in space.

The two main formalisms that exist to numerically solve the Navier-Stokes equations are the *Eulerian* and *Lagrangian* formalisms. Each formalism is named after the form of convection equation they use. The Eulerian formalism tracks the evolution of the independent variables, velocity and pressure, at fixed points in space. The Lagrangian formalism instead focuses on the change of velocity and pressure in space as these values are carried by markers that move with the flow. We will review these two formalisms, as well as the most common techniques affiliated with each method of determining a fluid's behaviour. Note that the Navier-Stokes equations can be converted such that vorticity (see section 3.2), instead of velocity/pressure, is the main field of concern. In short, Eulerian and Lagrangian methods can be used to track and update either or both forms of the Navier-Stokes equations.

An interesting side note is that the full solution to the unrestricted Navier-Stokes equations is one of the open Millennium Prize [36] problems in science today.

3.1.1 The Eulerian Formalism

The Eulerian method of solving the Navier-Stokes equations typically involve the discretization of space into a grid of cells. At each cell, variables of interest are stored and updated as a simulation runs. The most common discretization schemes used for the simulation of fluids in computer graphics are based on the *Marker-and-Cell* (MAC) grid specification for incompressible flow computations proposed by Harlow and Welsch [35].

At each cell in a MAC grid, the velocities are defined on cell faces, while pressure (and other scalars) are store at cell centers. Figure 3.1 illustrates the MAC grid specification for a cell at location (i, j, k) from a uniform grid discretization with the pressure, $p_{i,j,k}$, defined at the cell center, and the component velocities, $\left\{a_{i\pm\frac{1}{2},j,k}, b_{i,j\pm\frac{1}{2},k}, c_{i,j,k\pm\frac{1}{2}}\right\}$ defined at the cell faces [56]. The grid cell size is $\Delta x \times \Delta y \times \Delta z$.

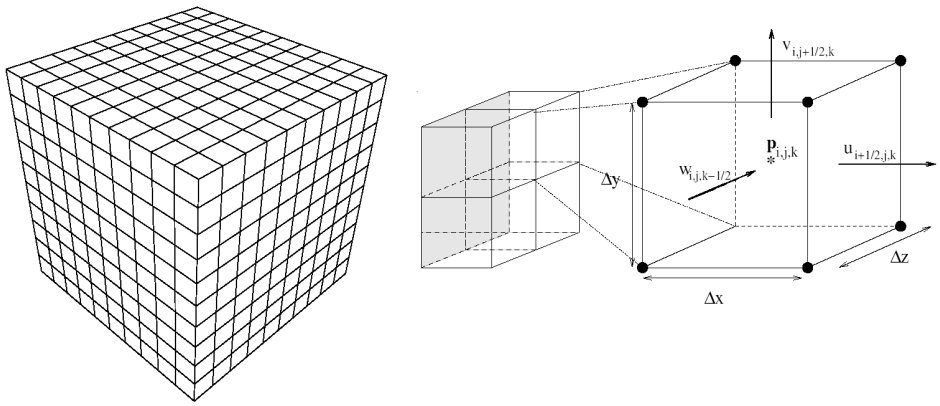


Figure 3.1: Uniform grid discretization ($left$) and MAC grid cell variables ($right$).

The component-wise velocities at the center of a cell can be approximated by taking the mean of the cell face velocities [35, 56]:

$$a_{i,j,k} = \frac{a_{i-\frac{1}{2},j,k} + a_{i+\frac{1}{2},j,k}}{2} \tag{3.4}$$

$$b_{i,j,k} = \frac{b_{i,j-\frac{1}{2},k} + b_{i,j+\frac{1}{2},k}}{2} \tag{3.5}$$

$$c_{i,j,k} = \frac{c_{i,j,k-\frac{1}{2}} + c_{i,j,k+\frac{1}{2}}}{2} \tag{3.6}$$

Stam's *Stable Fluids* paper [74] presents the basic process for solving the Navier-Stokes equations on a grid while maintaining the numerical stability of the simulation for an arbitrary time step size. Starting with the initial velocities at (the center of) each cell in the space discretizing grid, Stam devised a four step procedure for updating the velocity that maintains stability:

1. add external forces,

2. perform advection process,

3. perform diffusion process and

4. re-project temporary solution onto a divergence free field to maintain stability.

For each cell, the contribution of external forces are first added (assuming a fixed (i, j, k)):

$$a_{new} = a_{old} + \Delta t \vec{f}_{external} \tag{3.7}$$

$$b_{new} = b_{old} + \Delta t \vec{f}_{external} \tag{3.8}$$

$$c_{new} = c_{old} + \Delta t \vec{f}_{external} \tag{3.9}$$

Next, the effects of advection on the fluid are taken into consideration. The non-linear $\vec{u} \cdot \nabla \vec{u}$ term in 3.3 requires careful handling to maintain stability, and Stam's use of the *method of characteristics* [83] to ensure that stability is maintained regardless of the size of the time step used. This is a major contribution of the *Stable Fluids* work. Mathematically, the method of characteristics is a solution to advection equations of a particular form, however as Stam describes in [74], an intuitive description of this technique is easy to grasp and involves the tracing of particles backwards in time to determine their current locations in time. We will briefly describe this process below.

Previously, Foster and Metaxas [23, 24] used finite difference methods for handling the non-linearities in the Navier-Stokes equations; however, this resulted in stable simulations only for time steps bounded by the size of their grid cells and the inverse of the magnitude of the velocity. Stam instead traces the point currently at (i, j, k) backwards in time by $-\Delta t$ units through the current velocity field $\vec{u} = \{a_{new}, b_{new}, c_{new}\}$ and sets the new velocity at (i, j, k) as the linearly interpolated velocity at the backward-traced starting position. Figure 3.2 illustrates this novel backward-tracing advection update stage.

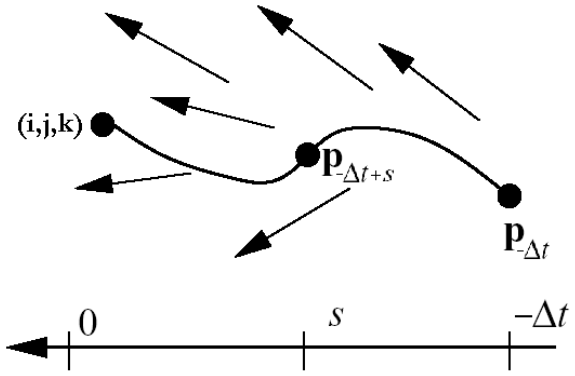


Figure 3.2: Tracing a position backwards in time to maintain stability [74].

Stam's fluid solver can also handle the effects of viscosity by solving $\frac{\partial \vec{u}_{new}}{\partial t} = \nu \nabla^2 \vec{u}_{old}$ using an implicit formulation that, after solving a sparse linear system, yields a solution that is stable for an arbitrarily large time step. The method of explicitly solving the diffusion equation (after discretizing the ∇^2 diffusion operator) as used by Foster and Metaxas becomes unstable when the viscosity is large [74].

The final and most significant step of Stam's simulator is the projection phase that converts the velocity field into a divergence free field. Stam uses a *Helmholtz-Hodge decomposition* method for determining the divergence free field. The Helmholtz-Hodge decomposition can segment any vector field, $\vec{\alpha}$ into the following form: $\vec{\alpha} = \vec{\beta} + \nabla h$, where β is a divergence free vector field and h is a scalar field. Using this decomposition, the projected field can be obtained by solving the following system:

$$\nabla^2 h = \nabla \cdot \vec{u}_{old} \tag{3.10}$$

$$\vec{u}_{new} = \vec{u}_{old} - \nabla h \tag{3.11}$$

Equations 3.10 and 3.11 require special attention to solve since equation 3.10 is a Poisson equation and solving it under the contraint of equation 3.11 adds an extra level of difficulty for current numerical algorithms used to solve these types of systems.

As with Foster and Metaxas' method, Stam's method could handle collisions of the

fluid with external objects while satisfying either the *no-slip* or *slip* boundary conditions. With slip boundary conditions[1], the normal component of the velocity is cancelled upon collision at the object's surface. With no-slip boundary conditions, the tangent component of the velocity is cancelled. Eulerian methods are especially well-suited for handling object interactions, since embedding an object into the space discretization scheme and performing cell-dependent collision detection and velocity modification is straightforward and physically plausible.

Stam's work provided a significant increase in effectiveness over previous techniques for simulating fluids in computer graphics, namely the work of Foster and Metaxas. The arbitrary stability and simple implementation of Stam's method made it the foundation for wide-spread use of similar techniques. The particle-tracking advection update step in Stam's work makes his method semi-Lagrangian, although it is widely accepted that the technique as a whole is Eulerian in formulation. This semi-Lagrangian method inherently introduces numerical dissipation into the solution, and Fedkiw *et al.* extend Stam's work by incorporating a technique from the computational fluid dynamics literature, known as *vorticity confinement*, to reduce this dissipation [19, 76]. Vorticity confinement is used to insert energy procedurally into a system to compensate for the apparent energy loss due to the use of the semi-Lagrangian tracking method. Mathematically, vorticity confinement abides by the laws dictated by the Navier-Stokes equations. The numerical dissipation manifests itself as a damping effect on the fine-scale detail of a fluid simulation. Fedkiw *et al.* re-inserted these details back into the simulation, thus maintaining a "lively" fluid.

A side contribution of the work by Fedkiw *et al.* is the introduction of a computationally inexpensive monotonic cubic interpolation scheme that replaces the linear interpolation used by Stam and summarized in equations 3.4, 3.5 and 3.6. Figure 3.3 illustrates some results from [19].

Another area of research for Eulerian solutions to the Navier-Stokes equations is the

[1]The slip boundary condition is sometimes referred to as the *no-through* boundary condition.

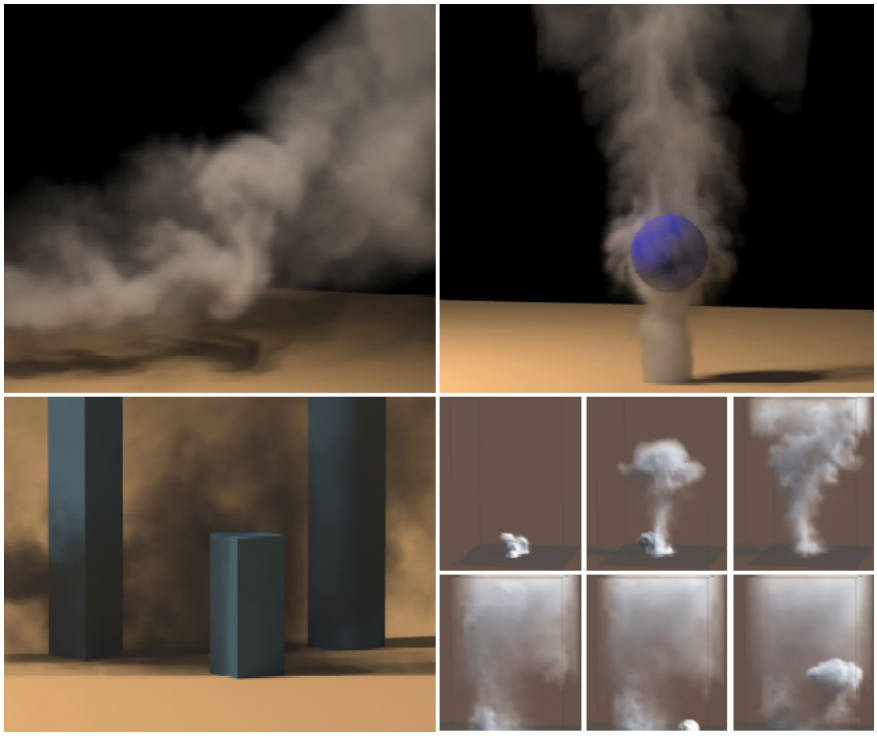


Figure 3.3: Fedkiw *et al.* inject energy back into the simulation to compensate for the numerical dissipation due to semi-Lagrangian particle tracing. This results in a fluid that maintains a more "lively" feel [19].

type of discretization used. The works so far have used uniform grid sampling. These methods do not employ biased importance sampling of flow effects in the generated fields. Losasso *et al.* propose an extension to the MAC grid that incorporates an octree data structure used to sample the flow behaviour according to importance and density [49]. A significant contribution of this work is the modification of Stam's update procedure to rectify the irregular volume sampling and gradient estimations; furthermore, Losasso *et al.* also expand their method to handle collisions across the octree nodes. Figure 3.4 illustrates the fine-level detail Losasso captured with an effective grid sampling of 1024^3.

In summary, Eulerian methods provide a simple, yet robust numerical solution to the Navier-Stokes equations for incompressible flow. Eulerian methods can handle arbitrary object collisions. Some drawbacks of Eulerian methods are that the sampling

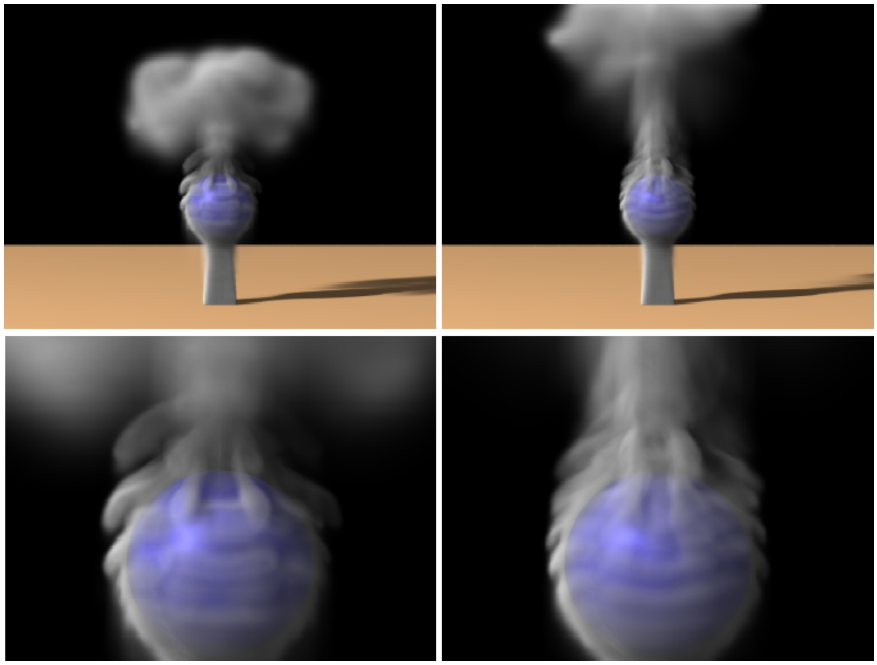


Figure 3.4: Fine level detail captured with an adaptive space discretization [49].

resolution used during discretization plays a large role in the ability to capture detailed flow behaviour, there is no straightforward solution to scale between coarse and fine discretization without re-simulating the flow[2] and Eulerian techniques scale according to the sampling density and not directly with the flow complexity. Interactive frame rates are only feasible for very coarse space discretizations, and simulations that capture complex flow behaviour with finer discretizations typically require time in the order of minutes to calculate one simulation step [74, 49, 19].

3.1.2 The Lagrangian Formalism

Unlike the Eulerian method of storing and updating the field values of interest (such as velocity, pressure and vorticity) at fixed positions in space using space discretization grids, the Lagrangian method updates the field values at abstracted markers in space.

[2]Since fluid flow is a chaotic phenomenon.

The underlying quantities carried by the markers is arbitrary. Some techniques use markers to represent the actual fluid particles and thus each marker would also carry a mass or density field function together with the flow field values of the Navier-Stokes equations. Other techniques use the markers as massless trackers that simply carry the flow; this generated flow can be used to induce movement on the actual fluid externally represented as a density field.

Lagrangian methods can theoretically focus the computation in explicit areas of interest, with marker sampling depending on the choice of marker representation. We will motivate the use of Lagrangian methods with a case study of the use of velocity/pressure Lagrangian techniques in computer graphics.

Smoothed Particle Hydrodynamics

Müller *et al.* presented a particle-based fluid simulator for computer graphics applications based on the heavily studied area of Smoothed Particle Hydrodynamics (SPH) from the CFD literature [50, 55]. Müller *et al.* base their simulator on the velocity/pressure form of the incompressible Navier-Stokes equations 3.1 and 3.2 with a Lagrangian method that tracks the evolution of these fields values at markers in space that carry flow behaviour as well as mass to represent particle-continuums in space.

This particle-based representation simplifies the complexity of enforcing the Navier-Stokes equations. Firstly, since the number of particles in the system remains constant throughout the simulation, and each particle has a fixed mass, then conservation of mass is automatically enforced. Secondly, the left hand side of equation 3.2 (or 3.3) can be simplified since the particles flow with the fluid. Thus, equation 3.2 can be rewritten as:

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{f} - \frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u}. \tag{3.12}$$

SPH estimation techniques are applied to particle systems to allow the evaluation of continuous field functions anywhere in space from the definition of these fields at discrete

locations in space (in the case of Müller *et al.*, the particle locations). The interpolation of a value, its gradient or its Laplacian from a set of discrete values are defined in SPH by (see [50, 55]):

$$I_c(\vec{x}) = \sum_i m_i \frac{I_i}{\rho_i} W(\vec{x} - \vec{x}_i; h) \tag{3.13}$$

$$\nabla I_c(\vec{x}) = \sum_i m_i \frac{I_i}{\rho_i} \nabla W(\vec{x} - \vec{x}_i; h) \tag{3.14}$$

$$\nabla^2 I_c(\vec{x}) = \sum_i m_i \frac{I_i}{\rho_i} \nabla^2 W(\vec{x} - \vec{x}_i; h) \tag{3.15}$$

where I_c, ∇I_c and $\nabla^2 I_c$ are the interpolated field quantities at location $\vec{x}$, m_i, ρ_i and x_i are, respectively, the mass, density and location of particle i, W is a smoothing kernel and h is the radius of influence of W. Müller *et al.* designed custom smoothing kernels that meet a simple set of constraints conducive to stability, such as zero values and vanishing derivatives at boundary locations [55].

Müller *et al.* proceed to address the contribution of pressure, viscosity and external forces, as well as an additional term modeling surface tension forces not present in equation 3.12. Rather than directly calculating these effects using the SPH equations (3.13, 3.14 and 3.15), Müller *et al.* modify these formulations to ensure symmetry and end up with the following force equations [55]:

$$\vec{f}^{pressure} = -\sum_i m_i \frac{p + p_i}{2\rho_i} \nabla W(\vec{x} - \vec{x}_i; h) \tag{3.16}$$

$$\vec{f}^{viscosity} = \nu \sum_i m_i \frac{\vec{u} + \vec{u}_i}{2\rho_i} \nabla^2 W(\vec{x} - \vec{x}_i; h) \tag{3.17}$$

Müller *et al.* update these forces as well as a physically motivated surface tension force every time step. After the force at each particle is calculated, leapfrog integration on the calculated acceleration updates the particle positions and velocities. Each particle is tested for object collision and reflected at a surface appropriately to handling boundary

conditions. Müller *et al.*'s technique yields interactive frame-rates for a non-trivial liquid simulation with 2200 particles. Figure 3.5 illustrates their results.

Figure 3.5: A Lagrangian fluid simulation with mass-carrying particles [55].

We have contrasted Eulerian and Lagrangian methods for simulating fluid flow using the velocity/pressure forms of the Navier-Stokes equations. We shall now elaborate on the alternative vorticity formulation of the Navier-Stokes governing equations as well as different types of vorticity representations, including the representation we use in our implementation.

3.2 Vorticity and Vortex Methods

Given a velocity field $\vec{u}$, the vorticity is defined as

$$\vec{w} = \nabla \times \vec{u}. \tag{3.18}$$

Vorticity can be intuitively thought of as an infinetely small whirlpool of circulation that spins about an axis perpendicular to the velocity at a point. The continuous culmination of all these whirlpools throughout the velocity field is the vorticity field. The curl of the Navier-Stokes momentum conservation equation 3.2 yields the *vorticity transport equation*:

$$\frac{\partial \vec{w}}{\partial t} + \vec{u} \cdot \nabla \vec{w} = \vec{w} \cdot \nabla \vec{u} + \frac{1}{\rho^2} \nabla \rho \times \nabla p + \nu \nabla^2 \vec{w} + \nabla \times F. \tag{3.19}$$

The vorticity formulation eliminates any dependence on a pressure term and automatically satisfies the continuity constraint [77]. A *vortex method* is simply a method based on the Navier-Stokes equations in vorticity form coupled with a Lagrangian tracking of the vorticity. Only vorticity and velocity information are required for these techniques. Cottet [16] noted that vortex methods are not susceptible to the same stability constraints as Eulerian methods since the non-linear convective term of the Navier-Stokes equation is not explicitly discretized for solving.

[57, 77] note that vortex methods can be faster than discrete Eulerian methods by up to an order of magnitude even under high vorticity flows. However, for flows with low viscosity, the fraction of the volume of flow regions with large vorticity and the flow regions without much vorticity is typically very small [77]. Thus, vortex methods can represent flow behaviour in a more compact fashion and we take advantage of this with our implementation.

The use of the vorticity field to simulate fluid flow has only recently come into mainstream attention in computer graphics with the works of [69, 4, 58]. Physical simulations in computer graphics sometimes make simplifying assumptions to the underlying model in hopes of gaining speed while minimizing any loss of visual accuracy. Since one of the primary objectives of our work to provide a faster physically motivated smoke simulator to artists, an analysis of the vorticity form of the Navier-Stokes equation could yield some insight into how it can be simplified for our purposes. Looking back at equation 3.19,

$$\frac{\partial \vec{w}}{\partial t} + \underbrace{\vec{u} \cdot \nabla \vec{w}}_{\text{Advection Term}} = \underbrace{\vec{w} \cdot \nabla \vec{u}}_{\text{Stretching Term}} + \underbrace{\frac{1}{\rho^2} \nabla \rho \times \nabla p}_{\text{Baroclinic Term}} + \underbrace{\nu \nabla^2 \vec{w}}_{\text{Viscous Diffusion Term}} + \underbrace{\nabla \times F}_{\text{External Forces}} \quad [77],$$

we shall, in what follows, investigate the impact each term will have on a simulation.

3.2.1 Advection Term

As with Eulerian methods, this term is responsible for the advection of the fluid from the flow. Sometimes a full time derivative of the vorticity is used to engulf both the change in vorticity and the advection term:

$$\begin{aligned}\frac{\mathrm{d}\vec{w}}{\mathrm{d}t} &= \frac{\mathrm{d}\ \vec{w}(\vec{p}(t),t)}{\mathrm{d}t} \\ &= \frac{\partial\vec{w}}{\partial t} + \frac{\partial\vec{w}}{\partial\vec{p}}\frac{\partial\vec{p}}{\partial t} \\ &= \frac{\partial\vec{w}}{\partial t} + \vec{u}\cdot\nabla\vec{w}\ ,\end{aligned}$$

where $\vec{w}(\vec{p}(t),t)$ is the Lagrangian formulation of the vorticity carried by a particle at position $\vec{p}(t)$ at time t. Needless to say, this term is vital in the flow behaviour and cannot be neglected. We will address advection in greater detail in later sections of this thesis.

3.2.2 Stretching Term

As vortices follow the flow, they stretch according to local deformations. Without these stretching effects, the flow would exhibit very robotic motion. The evolution of the flow is heavily affected by the interaction of vortices and certain interactions would not occur unless the vortex field is stretched to increase the area of influence. Hence, the stretching of vortices is an important flow mechanism that must be taken into account during simulation.

3.2.3 Baroclinic Term

In fluid simulations that require explicit surface tracking and large density and pressure variations, the baroclinic term is necessary to take these changes into consideration. One such example would be the simulation of a liquid/air interface, where the density

changes significantly at the interface. For the purposes of simulating smoke, we make the assumption that the density and pressure changes between the smoke and atmosphere are negligible. Thus, we omit the baroclinic term in the formulation of the vorticity transport equation we use for our simulation.

3.2.4 Viscous Diffusion Term and External Forces

Our simulation will make the inviscid assumption and set ν to zero, thus eliminating the viscous diffusion term. We explicitly handle external forces during the particle update procedure (see section 5.3).

3.2.5 Final Form

After the assumptions summarized above, we simplify equation 3.19 to the following form for our simulation purposes:

$$\frac{\mathrm{d}\vec{w}}{\mathrm{d}t} = \vec{w} \cdot \nabla \vec{u}$$

$$\frac{\partial \vec{w}}{\partial t} + \vec{u} \cdot \nabla \vec{w} = \vec{w} \cdot \nabla \vec{u} \quad .$$

3.3 Vortex Representation

A number of different vortex representations have been proposed for using vortex methods, as defined in section 3.2, to track the vorticity as a flow evolves. The choice of tracking representation plays an important role towards the organization and execution of a simulation, and different representations have different maintenance requirements.

We will briefly outline the four most common methods for discretizing the vorticity field, as summarized by [77], and later sections of this thesis will address our choice of representation in more detail. The four most common discretizations are *particles*, *sheets*,

volumes and *filaments*. Our model and implementation is based primarily on the vortex filament discretization approach.

Vortex particles where first used in a two-dimensional simulation by Rosenhead [67, 77] and in three dimensions by Chorin [12]. Vortex particle methods typically require very strict constraints on the spacing between vortex particles; particles' influence regions should ideally overlap, but inter-particle spacing must not grow larger than the radius of the influence. To take vortex stretching into account, the regions of influence of each vortex particle could be modified and extra vortex particles can be inserted or removed during simulation.

Vortex sheets are geometrically represented by line segments in two-dimensions or by planar regions in three-dimensions and the distinguishing trait of these methods is that explicit connectivity information over the line segment or planar regions must be maintained [77]. Re-meshing techniques are typically required in three-dimensional vortex sheet simulations to account for vortex stretching.

Vortex volumes have been used extensively in two-dimensional simulations [77], typically with triangles [26] or tetrahedra [31]. Vortex volume methods have not been employed often for three-dimensional simulations since the discretization and tracking of volumetric points, as well as inter-connected stretching enforcement are non-trivial.

Lastly, vortex filament approaches discretize vortices into closed loops of constant circulation (see section 3.6). Many techniques can be used to stretch and re-sample the loops, yet according to Kelvin's theorem the circulation for each filament need not be adjusted due to stretching effects [77, 68]. Since our method is based on vortex filament approaches, we will further investigate the rationale behind our choice of discretization in section 4.2 as well as the details to our specific implementation of this discretization in section 4.2.2.

3.4 Boundary Conditions with Vortex Methods

Handling collisions of a fluid against an external object using vortex methods is not a trivial process, as in Eulerian methods. Hess and Smith introduce the *panel method* in [45], Park and Kim apply the panel method to vortex fluid simulations for computer graphics [58]. The panel method can be used to satisfy the slip boundary condition by placing vortices on the boundary surface such that the slip condition (cancelling the normal component of the velocity at the boundary). Moreover, Park and Kim eject vortex particles slightly above the boundary surfaces to cancel the tangent component of the velocity at the boundary point to satisfy the no-slip boundary conditions. Figure 3.6 illustrates the use of the panel method and the modified emitting panel method for slip and no-slip boundary condition satisfaction.

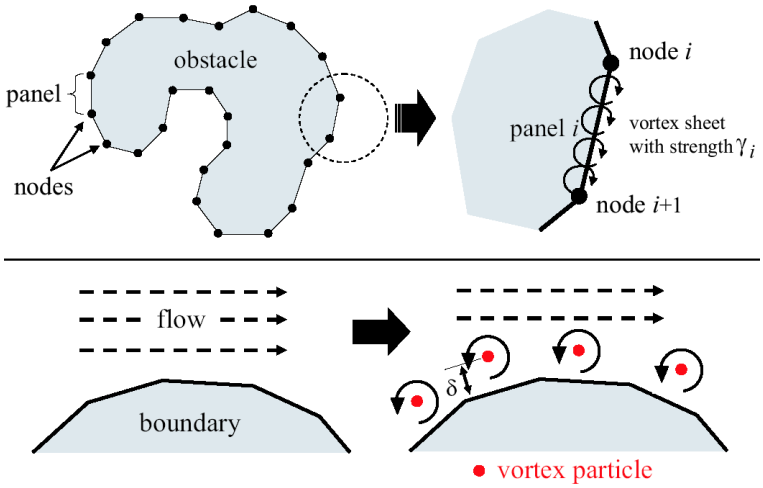


Figure 3.6: Park and Kim use the panel method (*top row*) and a modified ejecting panel method (*bottom row*) to satisfy the slip and no-slip boundary conditions using a particle vortex method [58].

Other methods have been proposed in computational fluid dynamics literature by Summers and Chorin [78] and Gharakhani and Ghoniem [29], however only the panel method has been employed for fluid simulations in computer graphics.

3.5 Velocity Field Reconstruction

Unlike Müller *et al.*, most vortex methods (and grid-based methods) do not couple the vorticity carrying objects, whether they be particles, filaments, volumes or sheets, with the fluid density. In this sense, these techniques simulate the evolution of the vorticity field and then use this field to displace fluid particles or densities which are in turn used to visualize the flow. This displacement cannot be generated directly from the vorticity field; a velocity field must be reconstructed from the vorticity field.

Independent of discretization, the velocity at a point $\vec{x}$ and time t generated by the flow captured by Lagrangian particles carrying vorticity $\vec{w}$ and influencing each other is defined by the *Biot-Savart Law* adapted from electromagnetic theory to the use of Lagrangian particles:

$$\vec{u}(\bar{x},t) = \frac{1}{4\pi}\iiint_P \frac{\vec{w}(\bar{p},t)\times(\bar{x}-\bar{p})}{|\bar{x}-\bar{p}|^3}\,d\bar{p}\ , \tag{3.20}$$

where P is the real domain of all vortices and $\vec{p}$ is the location of a single vortex. In practice, we discretize P and store vorticity at points in space. This equation can be used to compute the velocity at each discretized vortex position as well as the velocity at each visualization marker position. These velocities are used to displace the vortices and markers, after which stretching is taken into account to update the vortices. It is important to note that the Biot-Savart yielded velocity field is divergence free, and so we maintain the benefit of not having to explicitly enforce this requirement during our simulation [77].

Many techniques in computational fluid dynamics have been proposed to solve the Biot-Savart equation. The simplest method is direct integration of equation 3.20, however this technique has complexity

$$\mathcal{O}(N^2+NM) \;=\; \mathcal{O}(NM),\ N << M \quad ,$$

where N is the number of discretized vortices and M is the number of visualization markers. For large values of N or M, this method of calculating the velocity field is unacceptable for computer graphics simulations. Using a threshold distance cut-off is one way to reduce the complexity.

Christiansen [14] was the first to use a *vortex-in-cell* method to solve for the entire velocity field. First, a vortex field is generated by placing the vortices on a grid, and the velocity field is calculated by solving the following Poisson equation:

$$\nabla^2 \vec{u} = -\nabla \times \vec{w} \ . \tag{3.21}$$

The vortex-in-cell method requires that the vortex field be divergence free [77]. Vortex-in-cell methods can be coupled with direct integration techniques.

The *Fast Multipole Method* of Greengard and Roklin [32] extends the *Treecode Methods* of Barnet and Hut [8] that hierarchically subdivides vortices into clusters and uses multipole expansions to attain a theoretical performance of $\mathcal{O}(N)$ evaluating the Biot-Savart integral. Our method of evaluating the Biot-Savart integral is a hybrid direct-integration/hierarchal-caching algorithm and will be discussed in greater detail in section 4.2.3.

3.5.1 Mathematical Notes Regarding the Biot-Savart Formulation

Apart from the algorithmic challenges involved in efficiently utilizing the Biot-Savart formulation to determine the velocity field given a vorticity field, the formulation also suffers from some mathematical and physically-based considerations that we discuss here:

1. As noted in [1], the solutions to equation 3.20 are not unique since the field reconstructed after sampling all vortex and particle locations is divergence free, hence

any additional divergence free velocity field that satisfies the mass conservation and boundary conditions of the Navier-Stokes equations can be added to yield another valid solution. This is useful for adding external force fields.

2. The most notable mathematical disturbance of the Biot-Savart equation is the asymptotic singularity corresponding to local self-induction at $\vec{x} = \vec{p}$. Many theoretical filament models encounter this limitation. We will introduce a revised Biot-Savart formulation in section 4.2.3 that avoids this singularity.

3.6 Smoke Filaments

We chose vortex filaments as our discretization scheme for the simulation of smoke. For gaseous phenomena, vorticity tends to concentrate itself into filaments that can be used to compactly represent the flow field [51, 77]. Figure 3.7 are photographs of scientific studies in the visualization of vortex flow.

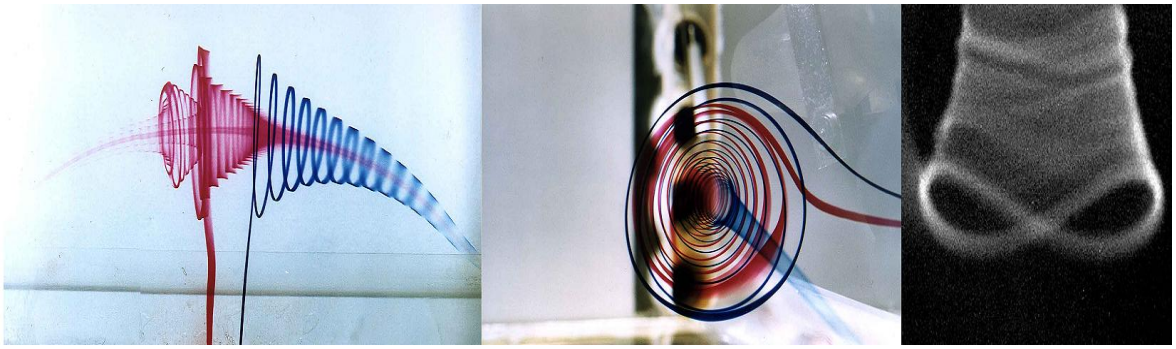


Figure 3.7: Vortex flow in fluids illustrating the concentration of vorticity into filaments [10].

The advantages of using the Biot-Savart law with Lagrangian curves carrying vorticity are the intrinsic incompressibility of the flow and the absence of numerical dissipation. Furthermore, vorticity structured in a loop curve can be simply understood: the structure

moves along the loop's axis and induces a gust of wind in that direction. Figure 3.8 is an illustration of our filament representation.

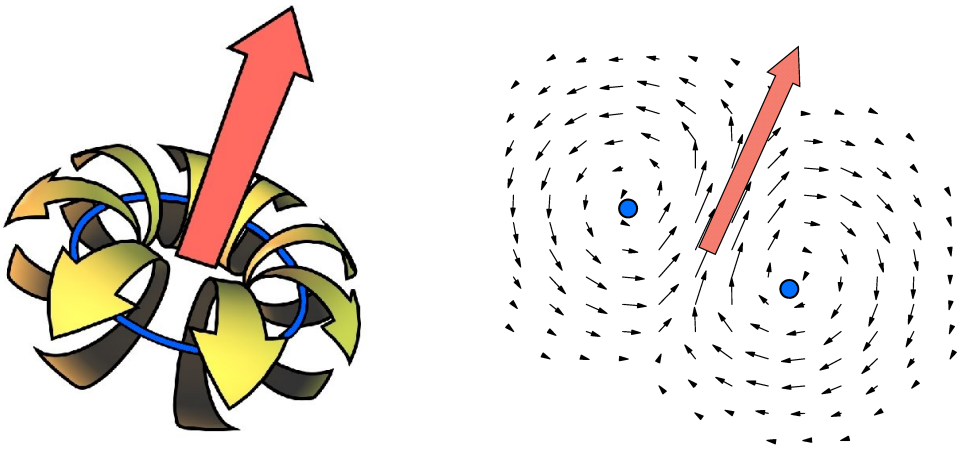


Figure 3.8: Left: A loop filament (blue) induces a gust of wind (yellow) traveling in the direction of the loop's axis (red). Right: a slice of the velocity field induced by the filament.

Defining loops is a convenient way to specify initial conditions and manipulate the fluid's motion. Thus our simulation is based on this primitive, although open curves may also be used in principle. Of the various ways to represent the geometry of a loop, we will propose a scheme appropriate for the simulation and control of smoke in section 4.2.2.

The consequences of using this discretization scheme are as follows:

1. The vorticity field is completely represented by the set of all filaments, and these filaments induce motion on themselves and surrounding filaments. Specifically, filaments induce their own motion (see Figure 3.9a) and filaments can interact to create complicated motion, such as the leapfrogging[3] shown in Figure 3.9b.

2. As long as we make the inviscid assumption, filaments have an infinite lifespan and conserve angular momentum in an interesting manner: the circulation remains

[3]Not to be confused with the leap frog integration scheme mentioned earlier in this chapter.

constant for each filament independent of time, due to Kelvin's theorem [9, 51], independent of time or stretching.

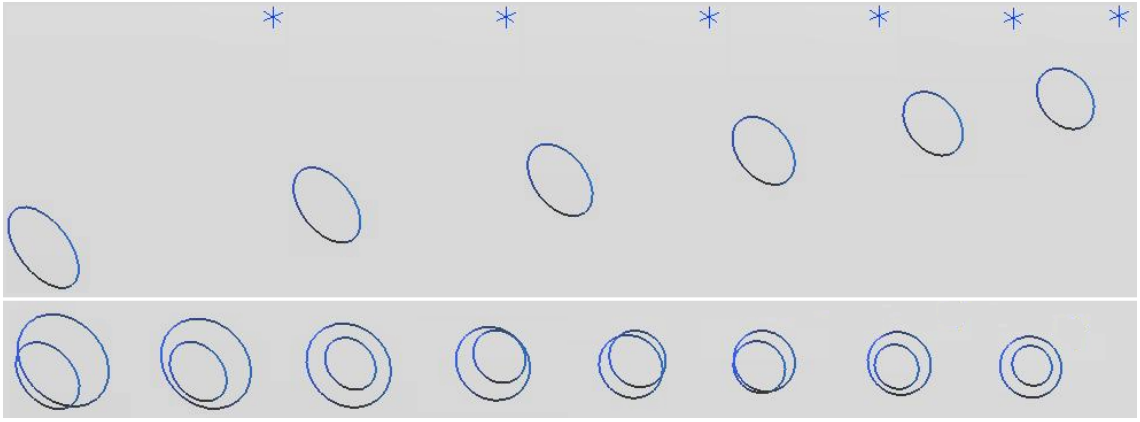


Figure 3.9: Our filaments induce their own motion (*top row*) and interact with other filaments, causing interesting filament behaviour such as the leapfrogging illustrated below (*bottom row*) [5].

The circulation of a filament, also sometimes referred to as its *strength*, is defined as:

$$\Gamma = \oint_L \vec{u} \cdot \vec{t}\, \mathrm{d}l = \iint_S \vec{w} \cdot \vec{n}\, \mathrm{d}s \quad [1], \tag{3.22}$$

where L is the border closed curve of a vortex tube cross section S, $\vec{u}$ is the velocity, $\vec{w}$ is the vorticity, $\vec{t}$ is the tangent to the closed-curve and $\vec{n}$ is the normal to the surface of the sheet. In practice, despite the infinite lifespan of a vortex filament, we optionally dissipate the strength to approximate atmospheric effects.

We will further elaborate on vortex filaments as well as our specific filament representation in section 4.2.2.

3.7 Summary

The Navier-Stokes governing equations of fluid dynamics were introduced and the two main formalisms used to solve these equations were contrasted. Examples of work in computer graphics using both these methods were overviewed and the benefits and shortcomings of each technique were discussed. An introduction to vortex methods, the method we chose to use, was followed by a review of different representations used in vortex methods and the details of performing simulations with these different representations.

Chapter 4

Control Techniques for Vorticity Based Flow

Chapter 3 focused on different methods of simulating fluids, including methods used in computer graphics applications. Although one type of application of fluid dynamics in computer graphics is concerned with the realistic and preferably physically-based *simulation* of fluid behaviour, another important application of fluid dynamics in computer graphics is the *animation* of a fluid's behaviour. The key distinguishing factor between these two areas is that the first is only interested in the output of a fluid simulation and the second is interested in manipulating the output of a physical simulation to introduce a controlled disturbance.

One of our main contributions is the ability to allow a user to interactively manipulate a smoke simulation, providing more flexible control over the behaviour of the smoke not previously available.

The problem of controlling the chaotic behaviour of a fluid is non-trivial. As with most physical systems, chaotic phenomena (such as smoke) behave in a non-linear fashion and prediction their motion, whether global or local, remains a difficult task. Furthermore, chaotic phenomena add an extra layer of difficult during prediction since they cannot be

controlled solely through the manipulation of the initial conditions and material properties of the system. Chaotic phenomena respond very sensitively to changes in initial conditions. Many different techniques have been employed to control a fluid in computer graphics.

We will first summarize previous work in the area of fluid control for computer graphics. Our method of control is novel since it abstracts control using a compact basis designed specifically to satisfy the requirements of physically-based simulation and computer animation. We will detail the basis and follow with the techniques employed to control our basis.

4.1 Previous Work in Smoke Control and Animation

During the early developments of fluid simulation in computer graphics (see chapter 2 and section 3.1.1), researchers paid scant attention to the potential for controlled animation. As the field of simulation in computer graphics matured, more serious questions were slowly being proposed regarding the coupling of control with simulation for fluids [24].

Foster and Fedkiw introduced one of the first attempts to control a fluid simulation in computer graphics [22]. Using a standard Eulerian grid-based Navier-Stokes simulator, Foster and Fedkiw allow a user to plant velocities in the grid in order to control the fluid at certain locations. This rudimentary form of control yields plausible results, however apart from explicitly enforcing the incompressibility of the fluid with the added velocities, this method of control does not obey the dynamics of the system.

In 2003, Treuille *et al.* introduced the first smoke simulator for computer graphics that allowed for a more regimented form of artistic control with the added benefit of generating an animation that respected the fluid dynamics of the system more precisely. Their method is based on *multiple shooting* techniques previously used in computer graphics to control rigid-body simulations [62].

Our work has similar goals as that of Treuille *et al.* as well as approaching the solution to these goals with similar motivations; as with Treuille *et al.*, we wish to provide an artist with *control* over the behaviour of smoke. Furthermore, our method of control and the method of Treuille *et al.* are both based on abstracting the low-level details of the control[1] mechanisms with tools that are familiar to artists. Treuille *et al.*'s choice of abstraction differ from our own (see section 4.3). Treuille *et al.* choose to use *key framing* to allow an artist to control a smoke simulation.

Their method allows a user to specify key framed shapes and uses an optimization procedure to apply the appropriate forces to the simulation such that the smoke forms shapes similar to the key framed shapes. The forces injected into the simulation are parameterized by a vector $\vec{h}$, and the simulation procedure is converted into the optimization problem of solving for the optimal parameter values [80]:

$$\text{argmin}_{\vec{h}}\ \phi_k(\mathcal{L}(\vec{q}_0, \vec{u})) + \phi_s(\mathcal{L}(\vec{q}_0, \vec{u})), \tag{4.1}$$

where $\mathcal{L}$ is the animation sequence derived from an initial state $\vec{q}_0$ and the applied forces $\vec{u}$, ϕ_k is the objective function that measures the deviation of the simulation output frames and the user key frames and ϕ_s is the objective function that is the sum of the applied forces $\vec{u}$ (used to minimize the amount of forces necessary to achieve the key framed behaviour, thus minimizing the potential for oscillations) [80]. Figure 4.1 illustrates some results from Treuille *et al.*.

They note that their technique suffers from scalability problems: if the scale of the regions of control are not on the same order as the global scale of the entire simulation, then the optimization procedure does not typically yield a sufficient solution. In other words, only large-scale detail can be controlled. Furthermore, their approach suffers from the classical numerical problems encountered during optimization, such as failing

[1] Unlike the work of Foster and Fedkiw.

Figure 4.1: Five balls of smoke rising to form the letters "SMOKE" [80].

to determine a global minimum of the objective function. As an introductory method of controlling smoke, key frame animation is useful. Furthermore, all of the multi-resolution simulation limitations outlined in section 3.1.1 still apply to this work since it is derived from a Eulerian grid-based solver. Lastly, this work is limited to two-dimensional simulations, and requires many simulation passes to reach the target animation sequence, making it an iterative and time consuming procedure. An artist has to wait a considerable amount of time before results are yielded. The method proposed in this thesis will allow for interactive control using intuitive user interaction handles and multi-resolution detail scaling.

In 2004, McNamara *et al.* extended the work of Treuille *et al.* through the use of an *adjoint method* for calculating the gradients necessary for an extended non-linear optimization problem similar to the one proposed by Treuille *et al.* [53]. McNamara *et al.* solve a force-constraint system similar to the system proposed in Treuille *et al.* However the physical simulation is now a fully three-dimensional simulation. They also present a set of control parameters specifically tuned to handle the control of free-surface liquids, as

well as gases [53]. Using the adjoint method for gradient calculation, McNamara *et al.*'s system is able to produce results for three-dimensional simulations faster than Treuille *et al.*'s system could for two-dimensional simulations, but a simulation still runs on the order of hours to days. Figure 4.2 illustrates some of the results from McNamara *et al.*

Figure 4.2: Key framed shapes animated in water and smoke [53].

As opposed to formulating the problem of controlling a smoke simulation as an optimization problem, Fattal and Lischinski instead choose to approach the problem from a unique and novel direction [18]. The *target driven* approach of Fattal and Lischinski uses a set of target smoke density states, instead of key frames, that serve the purpose of guiding the simulation towards a required goal. The significant contribution of this work is a reformulation of the Navier-Stokes equations that is specifically designed to incorporate the appropriate control forces, thus eliminating the need for expensive non-linear optimization of the simulation procedure [18].

In detail, Fattal and Lischinski outline their three main contributions. An additional *driving term* in equation 3.3 that guides the flow behaviour towards one of the user-specified target density states. A *gathering term* in the advection equation that minimizes the dissipative effects of the numerical solution, allowing the flow to match target densities more accurately. A system that allows for several smoke fields to be controlled within

close proximity. This contribution addresses another area of future work outlined by Treuille *et al.*

The *gathering term* is in place to handle transitions between density states that may require physically impossible operations, such as a transition between a low valued density field and a high valued density field. Although this term is not physically based, its motivations are similar to the motivation of vorticity confinement (see section 3.1.1), and restricting the dissipation of a simulation can allow for an interesting type of artist control not previously provided.

The *driving force term* is largely responsible for creating the motion required to match intermediate density states. Fattal and Lischinski re-formulate the Navier-Stokes momentum conservation equation as:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{f} - \frac{1}{\rho}\nabla p + \alpha \mathbf{F}(\rho, \rho) + \beta \vec{u} \quad [18], \tag{4.2}$$

where $\mathbf{F}$ is the force that drives the density field from its current value, ρ, to the next target value, ρ and the two constants α and β scale the system to conserve momentum. Fattal and Lischinski show that after blurring the original density to avoid numerical issues in regions of zero density, the driving force can be solved:

$$\mathbf{F}(\rho, \rho) = \nabla \rho' \tag{4.3}$$

where ρ' is the blurred target density state [18]. It should be noted that the method of Fattal and Lischinski, due to its simplicity, operates much faster than the methods of [80] and [53]. However, since it is based on a grid-based Eulerian solver, simulation times still span on the order of tens of minutes to hours. Figure 4.3 illustrates some of their results.

As with the work of Losasso *et al.*, Feldman *et al.* present a Eulerian grid-based data structure that allows the coupling of structured tetrahedral meshes with a standard

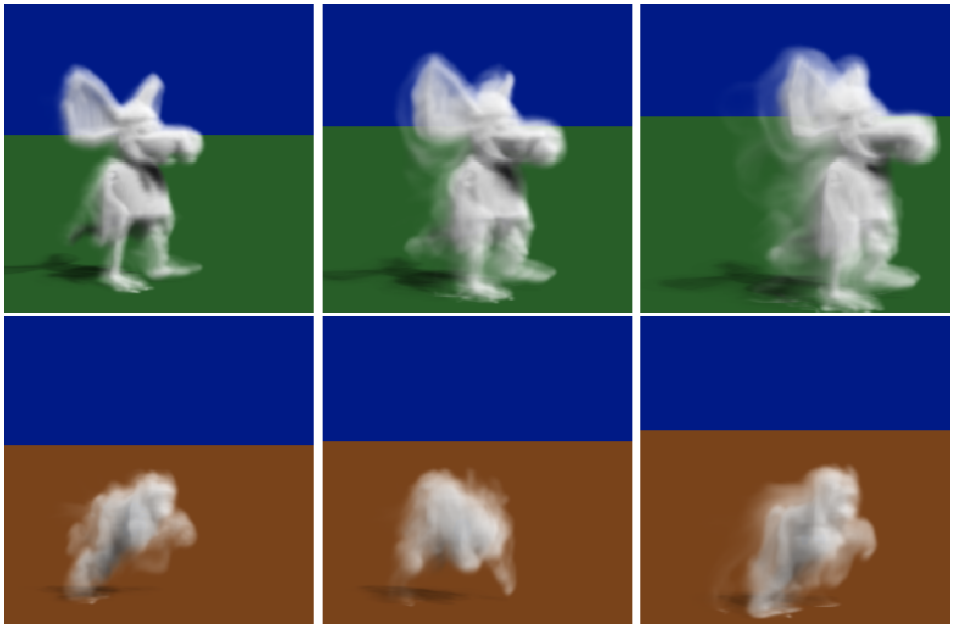


Figure 4.3: Note the sharp details due to the gathering term of Fattal and Lischinski's method [18].

discretization grid [20]. This allows artists to simulate the complicated flow of gases with various geometric shapes. These shapes can be used in turn to control the behaviour of smoke.

Although Feldman *et al.* present novel solutions to the standard issues involving a new Eulerian discretization method, such as interpolation across discretized cells, mass conservation and the semi-Lagrangian integration scheme, their method suffers in performance relative to standard and octree accelerated Eulerian solvers [20]. The results generated are very impressive, but their system only handles static tetrahedral meshes and there does not seem to be any proposed solution for a more robust and expressive artistic control system. Figure 4.4 illustrates some results.

In summary, previous methods for controlling smoke and fluid animations can yield stunning results, yet an artist cannot use any of the current techniques to edit the flow of smoke in real-time while maintaining realistic flow behaviour. We will introduce our

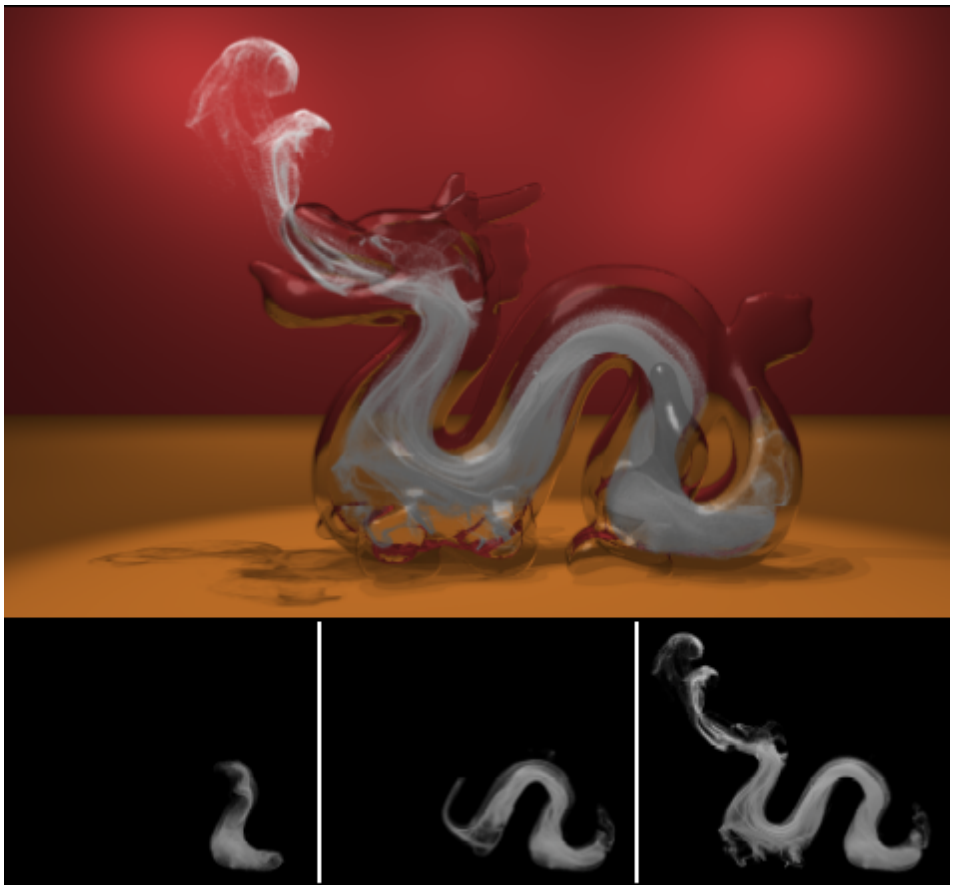


Figure 4.4: A smoke simulation exhibiting complicated object collision (*top row*) used to control the behaviour of the smoke (*bottom row*) [20].

technique for controlling smoke by first discussing the underlying basis of representation. This basis is coupled to both the simulation and control aspects of our system. Allowing artists to use animation tools with which they are already comfortable and in a familiar workflow are the open problems we address in this work. Our vortex method representation will be discussed in the following sections.

4.2 Multi-resolution Filament Basis

Our system is a *vortex method* based smoke simulator. We recall from section 3.2 that a vortex method is a solution to the vorticity form of the Navier-Stokes equations coupled

with a Lagrangian tracking method of the vorticity field. Our choice of tracking method is based on vortex filaments (see section 3.3) which, among other benefits, maintain constant circulation and provide a meaningful geometric link to the vortex stretching operation. Recall that the $\vec{w} \cdot \nabla \vec{u}$ term in equation 3.19 describes the stretching of the vorticity field through advection.

4.2.1 Previous Vortex Filament Representations

A naïve method to represent a filament is a piecewise linear approximation to a closed curve, composed of point samples connected with line segments. During simulation, filaments evolve through stretching and other deformations. Thus the approximation quickly becomes undersampled. Resampling each filament to accurately represent the increasingly complicated filament curve becomes difficult [4, 1, 5].

Angelidis and Neyret present a high-resolution filament sampling and level-of-detail scheme in [4] that addresses some of the issues regarding the piecewise linear method of representation. Starting with a coarse sampling (two points and one segment for an open curve and three points and three segments for a closed curve), a level-of-detail binary tree is constructed for every approximated curve with segments stored at tree nodes. The leaves of the tree are the segments of the highest-detail curve, and each non-leaf node is a geometric average its two children nodes[2]. During the first iteration, the only parent node is the root node [4]. The purpose of the level-of-detail tree is to simplify the approximate calculation of the velocity field generated by the vorticity being carried by the filament curves.

During filament advection, the point samples on the approximated curve are advected and a new level-of-detail binary-tree is constructed with the updated curve. Segments are sub-divided if they are stretched past a certain threshold. Since evaluating the velocity at a point $\bar{p}$ is normally an expensive $\mathcal{O}(N^2)$ operation (with N being the sum of curve

[2]The approximated *whirl* of a parent node must equal the sum of the whirls of its children [4]

samples across all filaments in the simulation), a computationally feasible approximation of the velocity is calculated using the level-of-detail tree and an adaptive error metric.

Our method of representing a vortex carrying filament avoids the sampling issues presented above by modeling a filament mathematically as a periodic signal. Our representation is based on the periodic signal processing literature. We will elaborate on how we employ a novel three step projection mechanism to control the degree of complexity and sampling of our vorticity carrying filaments. Furthermore, our representation guarantees the stability of our numerical solution for *arbitrarily long* time step durations. During each time step, all filaments are synthesized and advected. We will present the details of filament advection as it pertains to our representation after elaborating on our multi-resolution filament basis representation.

4.2.2 Filament Co-ordinate Frame Analysis

We present a mathematical representation for vortex filaments that specifically supports multi-resolution filament detail. Our representation is derived using a method similar to the covariance method used to generate a principal component analysis of a system [38]. During simulation and animation, an artist can control the degree of filament complexity by controlling the number of harmonic coefficients used to reconstruct each filament. This allows multiple levels of editing, starting from coarse to fine level physically modeling analogous to the way some painters paint in layers: adding detail to the preceding layer with each new layer until the final image is created. This type of multi-resolution editing of fluid behaviour has not been available with Eulerian grid-based simulations[3].

We perform the principal component analysis and Fourier series decomposition of a filament's periodic signal on the following initial representation of a filament: a filament of length L is initially defined with n point samples $\{c_i | i = 1, 2, \cdots n\}$. The geometric

[3]For example, performing an Eulerian fluid simulation at one grid resolution, then simply rescaling the grid resolution, will not yield coherent physical behaviour.

centroid of the points is defined as

$$\bar{c} = \frac{1}{n}\sum_{i=1}^{n} c_i \ . \tag{4.4}$$

Furthermore, we define $\mathbf{c}(l)$ as the cyclic linear interpolation of the c_i's. A suitable filament coordinate frame is built such that a Fourier series decomposition of the signal can be conducted: the 3×3 covariance matrix of the c_i points is defined as:

$$\frac{1}{n}\sum_{i=1}^{n}(c_i - \bar{c}) \cdot (c_i - \bar{c})^{\mathrm{T}} \ , \tag{4.5}$$

and the eigenvectors of this matrix, $\lambda_x > \lambda_y > \lambda_z$, are the orthogonal unit vectors that, with the centroid $\bar{c}$, define the local coordinate frame of the filament. These unit orthogonal eigenvectors, $\{\vec{e}_a | a \in [x, y, z]\}$, also define a planar approximation $(\bar{c}, \vec{e}_z)$ to the closed curve that is the best fit to the filament's geometry. Given a filament's local coordinate frame, the periodic signal describing the filament is defined as [5]:

$$s(\theta) = \begin{pmatrix} \vec{e}_x \\ \vec{e}_y \\ \vec{e}_z \end{pmatrix} \cdot \left(\mathbf{c}\left(\frac{L\theta}{2\pi}\right) - \bar{c} \right) \ . \tag{4.6}$$

The Fourier series representation of each of the dimensions of the 2π-periodic signal of the filament is determined such that $s(\theta)$ can be reconstructed with an arbitrary amount of precision depending on the number of harmonic coefficients used. In the limit,

$$s(\theta) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \ cos(n\theta) + b_n \ sin(n\theta) \right) \ ,$$

where a_n and b_n are the Fourier coefficients, defined as:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(\theta) \ cos(n\theta) \ d\theta \tag{4.7}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} s(\theta) \ sin(n\theta) \ d\theta \ \ . \tag{4.8}$$

The reconstructed signal, $s'(\theta)$, depends on the number of harmonic coefficients. These can be controlled by an artist during animation to trade off the speed and accuracy of the simulation. An important note is that although using more coefficients to

reconstruct a filament yields a more accurate simulation, this accuracy typically presents itself in high-frequency detail, as opposed to large scale variation. This means that an artist can model and animate our smoke using very few coefficients to reproduce the filament signal, and once modeling is complete, a large number of coefficients can be used to simulate and render the final animation sequence without losing any of the large-scale flow behaviour modeled at the lower resolution.

Given the reconstructed signal approximation to the complete filament signal, a newly synthesized curve can be obtained as follows [5]:

$$\mathbf{c}'(l) = \bar{c} + \begin{pmatrix} \vec{e}_x & \vec{e}_y & \vec{e}_z \end{pmatrix} \cdot s'\left(\frac{2\pi l}{L}\right) \ . \tag{4.9}$$

Figure 4.5 illustrates our filament representation. Figure 4.6 shows the evolution of four identical simulations, each using a different number of coefficients to reconstruct the filament signals. For each filament, the centroid $\bar{c}$ and basis vector $\vec{e_z}$ can be used to identify the *average location of motion* and *approximate direction of motion* for that filament.

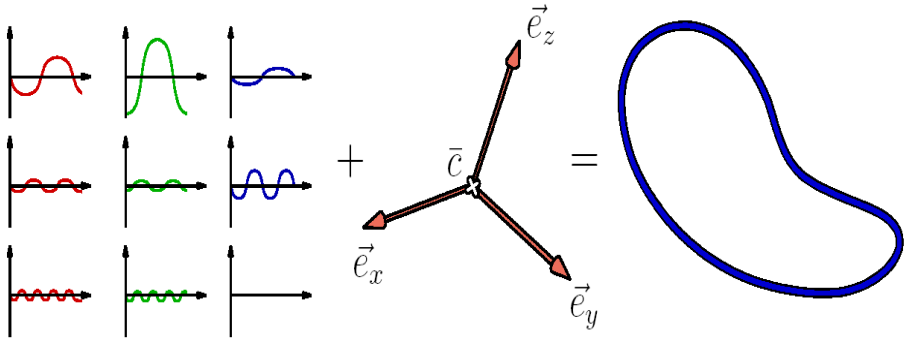


Figure 4.5: The parametric description of a filament's geometry (blue) consists of a local coordinate system (red) and three Fourier series (shown on the left as curves).

The advection of the filaments will be described below, followed by an elaboration on the suitability of our basis for controlling our smoke simulation, as well as the mechanisms

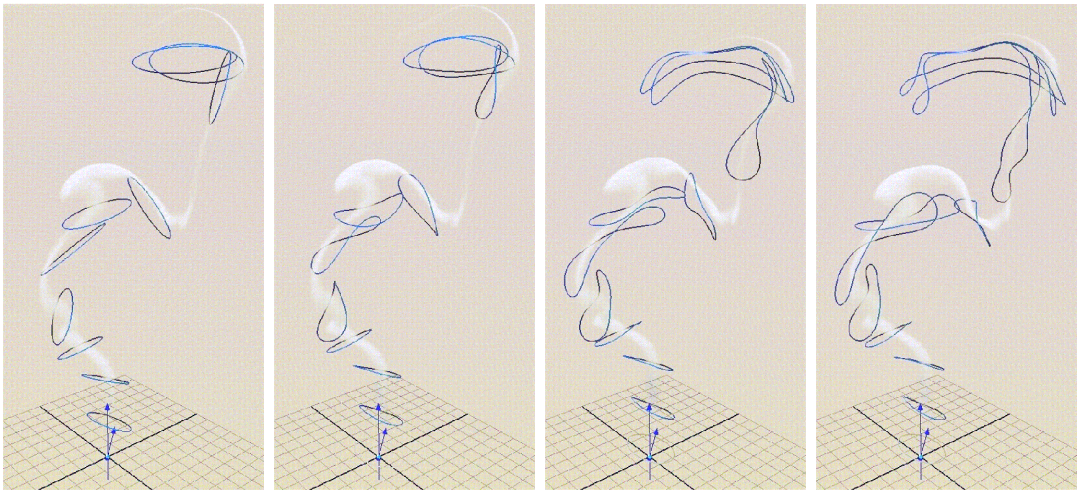


Figure 4.6: Simulation of filaments with bounds on the different number of frequencies (i.e., increasing bandwidths). From left to right: 1, 2, 4 and 8. Note the similarity in the motion.

we use to control our basis.

4.2.3 Advection of Filaments

During every time step of our simulation, a three step procedure is required to update each filament. *Geometry synthesis* applies equation 4.9 to the current parameters of the filament using the user-controlled number of coefficients to reconstruct the filament. *Advection* reconstructs the (approximate) velocity field from the vorticity field carried by all the filaments in the system and then advects the filament points with the field [4]. We will elaborate on the methods used to generate the velocity field and advect the filaments in this section. *Harmonic analysis* reconstructs a new Fourier representation of the updated/advected filament using the procedure outlined above.

To reconstruct the velocity field from the vorticity field, the Biot-Savart law was

[4]Smoke particles used to visualize the flow and represent the actual smoke are also advected by this field, however their advection and visualization will be covered in chapter 5.

introduced in section 3.5. The numerical and algorithmic issues involved with evaluating the velocity at a point using the Biot-Savart law were also introduced and we shall now discuss these issues in further detail.

Modified Biot-Savart Kernel and Integral Evaluation

If, using the *filament assumption*, we assume that vorticity is only concentrated at the filaments, we can re-write the Biot-Savart equation for the velocity at a point due to *one* filament, c'(l), with $l \in [0, L]$, as

$$\vec{u}(\bar{p}) = \int_0^L \mathcal{S}\ K_{BS}\left(\frac{|\bar{p} - \mathbf{c}'(l)|}{r}\right)(\bar{p} - \mathbf{c}'(l)) \times \vec{\tau}(l)\ \mathrm{d}l \tag{4.10}$$

where K_{BS} is the spatial weighting kernel, S is the circulation (or *strength*) of the filament, r is the thickness of the filament and $\vec{\tau}$ is the cyclic piecewise linear interpolation of the tangent of the filament at the corresponding point. Of special numerical importance is the weighting kernel. We designed our kernel to closely represent the physically correct distance fall-off rate, while also satisfying continuity over its whole domain and performance-efficient computation. The kernel we have implemented is defined as

$$K_{BS}(x) = \begin{cases} (4 - \frac{20}{x^2+4})^2 & \text{if } x^2 < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

Furthermore, this kernel is monotonic, C^1 continuous, numerically very close to C^2 continuous and almost anti-symmetric about $\frac{1}{2}$. Figure 4.7 illustrates our kernel's response. Although simpler polynomial kernels may be used, we have found in practice that this particular kernel yields pleasing results and requires few arithmetic operations to implement.

The Biot-Savart integral can be evaluated as a Riemann sum using the samples c_i or using the connected edges[5]. Once the velocity induced by *one* filament at each particle

[5] [7] states that using edges as the discrete domains will help avoid motionless areas around a filament with small thickness.

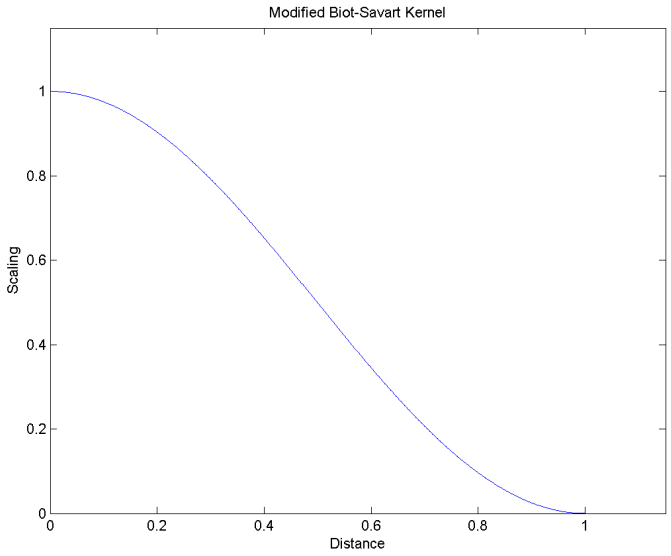


Figure 4.7: The distance fall-off profile of our modified Biot-Savart kernel [5].

location is calculated, the total velocity at each particle location is merely the sum of the individual contributions of the filaments.

We note that there is an underlying geometrical connection between the Biot-Savart law and the *twist* (also sometimes referred to as the *screw*) transform that we can take advantage of for higher-order advection of filament and smoke particles. We will elaborate on the geometrical connection that allows us to obtain a higher-order advection method as well as another reformulation of the Biot-Savart law that uses this connection.

Higher-order Advection Method

The Biot-Savart law can be explained in physical terms as being responsible for determining the velocity induced at a point due to a vorticity field, it can be also be investigated in geometric terms as determining the velocity induced on a point due to simultaneous rotations proportional to the Biot-Savart integrand kernel (see section 4.2.3 above) [5, 4]. This velocity is tangent to the trajectory of the particle and the method described in section 4.2.3 can be used to approximate the new trajectory with a linear translation of

the form $\bar{p} + \Delta t \cdot \vec{u}(\bar{p})$, where the velocity is determined using our modified Biot-Savart kernel and Δt is the time step.

A *twist* transform is a transformation composed of multiple simultaneous rotations [3, 27] and is formulated as a transformation with translational and rotational components. The original Biot-Savart law can be reformulated as the product of a matrix composed of a scaled twist transformation and the input position:

$$\vec{u}(\bar{x}, t) = \left[\iiint_P \frac{1}{4\pi \ (|\vec{x} - \vec{p}|^3)} \ \mathbf{T}_{\vec{w}}^{\bar{p} \times \vec{w}} \ d\bar{p} \right] \cdot \bar{x} \quad , \tag{4.12}$$

where the twist transform is defined as

$$\mathbf{T}_{\vec{w}}^{\bar{p} \times \vec{w}} = \begin{pmatrix} 0 & -w_z & w_y & p_y w_z - p_z w_y \\ w_z & 0 & -w_x & p_z w_x - p_x - w_z \\ -w_y & w_x & 0 & p_x w_y - p_y w_x \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \vec{w} \times & \bar{p} \times \vec{w} \\ 0 \ \ 0 \ \ 0 & 0 \end{pmatrix} \quad [5, \ 4].$$

Considering this formulation gives us more insight into our choice for a Biot-Savart kernel, since it is clear now that the kernel describes the scaling of the simultaneous rotation velocities as a function of distance from the point they are being applied to.

Let us define the term between the square brackets of equation 4.12 as the matrix $\mathbf{M}$. We can think of the *exponential of a matrix* as performing an integration operation on a transformation's motion path. The *logarithm of a matrix*, considering only matrices that perform rigid-body transformations, is a derivative of the motion path defined by the transformation. In other words, taking the logarithm of a matrix yields the tangent to its motion path. Since M encodes the scaled simultaneous rotations (*twists*) applied on a point $\bar{p}$, we can advect $\bar{p}$ in the usual fashion with $\bar{p'} = \bar{p} + \Delta t \cdot \underbrace{\mathbf{M} \cdot \bar{p}}_{\vec{u}(\bar{p})}$ **or** we can extract the trajectory of $\bar{p}$ using the matrix exponential operator:

$$\bar{p'} = exp(\Delta t\ \mathbf{M}) \cdot \bar{p} \quad , \tag{4.13}$$

given that $\mathbf{M}^{(t)} = exp(t\ \log \mathbf{M})$. The full expansion of the exponential operator is defined in [27] as

$$exp(\mathbf{T}_{\tilde{\mathbf{w}}}^{\bar{\mathbf{p}}\times\tilde{\mathbf{w}}}) = \begin{cases} \mathbf{I} + \mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}} & \text{if } |\vec{w}| = 0 \\ \mathbf{I} + \dfrac{1 - cos(|\vec{w}|)}{|\vec{w}|^2}(\mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}})^2 + \dfrac{sin(|\vec{w}|)}{|\vec{w}|}\mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}} & \text{if } \vec{w}\cdot(\bar{p}\times\vec{w}) = 0 \\ \mathbf{I} + \mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}} + \dfrac{1 - cos(|\vec{w}|)}{|\vec{w}|^2}(\mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}})^2 + \dfrac{|\vec{w}| - sin(|\vec{w}|)}{|\vec{w}|^3}(\mathbf{T}_{\vec{w}}^{\bar{p}\times\vec{w}})^3 & \text{otherwise.} \end{cases}$$

We can see that the first term of the expansion encodes the linear trajectory used prior to the introduction of the higher-order advection method. The equation for the discretely integrated twist trajectory induced by a single filament on a point using the modified Biot-Savart kernel introduced in section 4.2.3 is

$$M = \sum_{i=0}^{n} \mathcal{S}\tilde{f}(\|\bar{p} - c_i\|^2/r^2) \begin{pmatrix} \vec{\tau}_i \times & c_i \times \vec{\tau}_i \\ 0\ \ 0\ \ 0 & 0 \end{pmatrix} \tag{4.14}$$

Figure 4.8 illustrates the difference between linear advection using the generated velocity field and our higher-order advection method.

After the appropriate twist transforms are calculated, advecting the vortex filaments (and smoke particles) with the higher-order advection procedure can be performed in parallel. Section 6.4.2 will elaborate on a Graphics Processing Unit (GPU) advection acceleration component we have implemented.

4.2.4 Filament Stretching During Advection

After a filament is advected and re-analyzed, the total kinetic energy of the system must be preserved (to satisfy the vorticity stretching term in equation 3.19: $\vec{w} \cdot \nabla \vec{u}$).

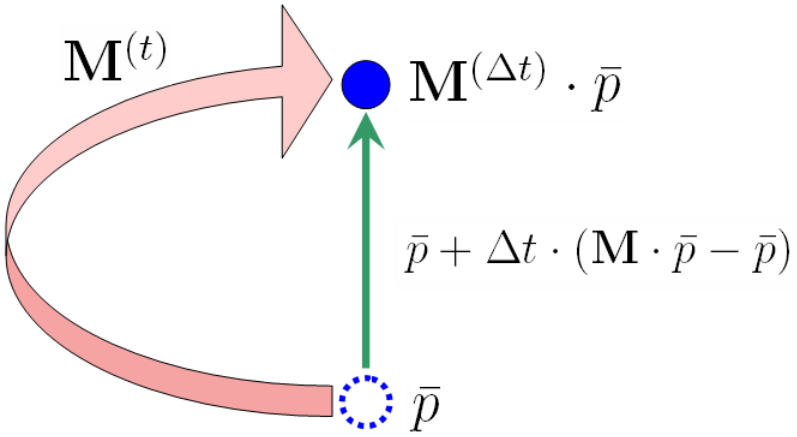


Figure 4.8: Comparison between linear and higher-order advection methods used on vortex filaments and smoke particles [5, 4].

The stretching term can be enforced by preserving the strength of each filament after deformation [6].

As a filament stretches, its strength must decrease and we enforce this using a simple ratio in the difference of lengths between the previous filament state and the current filament state. The new filament strength, after stretching is:

$$\mathcal{S}' = \frac{L}{L'}\mathcal{S} \quad [5]. \tag{4.15}$$

Although in real physical systems a vortex filament may stretch indefinitely, we clamp the maximum length of a filament in our simulation in order to maintain the support of the flow generated by many subsequent filaments. Each filament in our simulation also has a lifespan. Alternatively, we could link the lifespan of a filament to a constraint on its (maximum) length.

So far we have summarized some key details on the flow carrying physical nodes in our simulation. However, we have not yet elaborated on how we control the overall flow. We

[6] Kelvin's theorem states that the circulation remains constant for inviscid (and barocyclic) flow as well as linking the preservation of circulation to the stretching term in the vorticity form of the Navier-Stokes equation [5, 68].

will present the control mechanisms used together with our multi-resolution basis to control the behaviour of each filament while maintaining a realistic and physically motivated simulation behaviour. Furthermore, the abstracted control tools we have implemented for an animator to control the smoke's behaviour *in real-time* using our system will be presented below.

4.3 Filament Basis Control

Earlier in this chapter we presented our novel multi-resolution basis used to describe the vortex filaments that carry the flow behaviour of our system.

To manipulate the direction of the flow, filaments must be guided towards the desired direction without disturbing the differential flow trajectory. Simply repositioning filaments in hopes of shifting the region of flow support will not yield the desired behaviour. Alternatively, we modify the parameters of a filament such that the desired flow is self-induced by the physical simulation which maintains consistency. We will present two mechanisms we have used to modify the filaments such that they naturally induce the desired motion in the following section.

4.3.1 Control Mechanisms

The two underlying mechanisms used to control the direction of the flow are *paddling* and *turning*.

Paddling

Using the Biot-Savart law as a reference, we see that according to the distance fall-off kernel, the movement induced on a filament due to its own influence is typically greater than the influence on the filament by other filaments in the system, assuming that the strength of filaments in the system are of approximately the same magnitude. Thus,

we can reweight a filament's strength such that the self-induction steers that filament in a desired direction. This behaviour is physically plausible and generates motion that maintains the flow support criterion.

The filament strength is redistributed according to the following profile function [5]

$$\gamma(c_i) = \vec{m} \cdot \vec{\tau}(c_i) \quad , \tag{4.16}$$

where $\vec{m}$ is the desired direction of motion for the filament and τ is the tangent at the filament redistribution locations. The profile function $\gamma(c_i)$ is bounded in the interval $[0, 1]$ and conserves the total kinetic energy of the system since

$$\int_0^L \vec{m} \cdot \vec{\tau}(c_i) \, \mathrm{d}l = 0 \; .$$

Energy about a filament is merely being shifted to the locations it is required in order to induce the proper motion and the circulation about each manipulated filament remains constant. For more details please refer to [5].

To simulate the effects of an external force, such as a gust of wind, acting on the filament, we multiply the strength of a filament by $1 + f_p\gamma(c_i)$, where f_p is the amount of paddling. This increases the vorticity in the direction of $\vec{m}$ and weakens the vorticity in all the other directions. Figure 4.9 is a conceptual illustration of the effects of paddling.

Turning

If the strength of the filaments in a system are unevenly distributed, or the desired rate of change of the flow direction is large, the self-induced motion change of the paddling method might not be adequate. In this case, we can rotate a filament's frame by rotating its average direction vector $\vec{e_z}$. We call this mechanism *turning* and although the changes it causes to the flow behaviour are not physically motivated, it introduces discontinuities

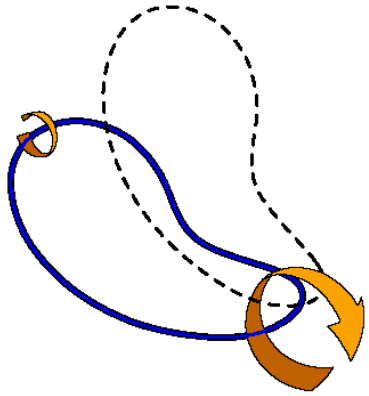


Figure 4.9: Paddling allows a filament to induce motion in a desired direction by modifying the magnitude of the vorticity (*curly arrows*) at the appropriate locations around the filament [5].

to the flow behaviour that are not as extreme as the naïve approach of frame shifting explained above.

Since turning does not maintain the flow support criterion, paddling is the preferred mechanism of control. Paddling will create motion that is physically based, yielding animations that behave much like real smoke would. The amount of turning used to control a flow is a user controllable parameter since, if used excessively, it will yield results that may look suspicious to an experienced critic. Figure 4.10 illustrates turning's frame rotation effect.

4.3.2 Control Tools

Some of the more rudimentary fluid control methods mentioned earlier in this chapter allow a user to add control to a simulation by adjusting physical parameters. This method of control is frustrating for a user since the effects of parameter tuning, especially when applied to chaotic phenomena, are hard to predict and because grid-based simulators require a large amount of time before results can be generated. Our system aims to

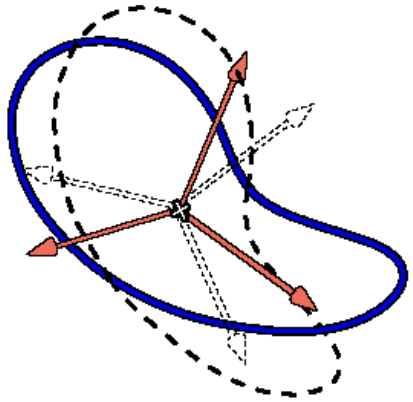


Figure 4.10: Turning modifies the direction of a filament by rotating its local frame [5].

solve this workflow problem by reducing the turn-around simulation time (often yielding simulations that are run in real-time or interactive frame rates) as well as supporting a multi-resolution level of animation and detail control that scales intuitively from an artist's perspective. Furthermore, the simulation is controlled by tools with which artists are familiar with as opposed to tweaking, for example, the amount of paddling and turning. We abstract these low-level control mechanisms away with tools that provide intuitive and predictive flow manipulation.

This section will describe the three control tools we have created for meaningful and artistic smoke simulation control: *current control curves*, *current control attractors* and *current tornado effects* [5].

Current Control Curves

The extensive use of two-dimensional curves embedded in three-dimensional space in the areas of computer animation and modeling makes them a familiar tool for computer artists. Figure 4.11 illustrates the use of curves for various purposes in a few graphical content creation packages.

Controlling the flow carrying filaments leads to indirect control of the smoke particles

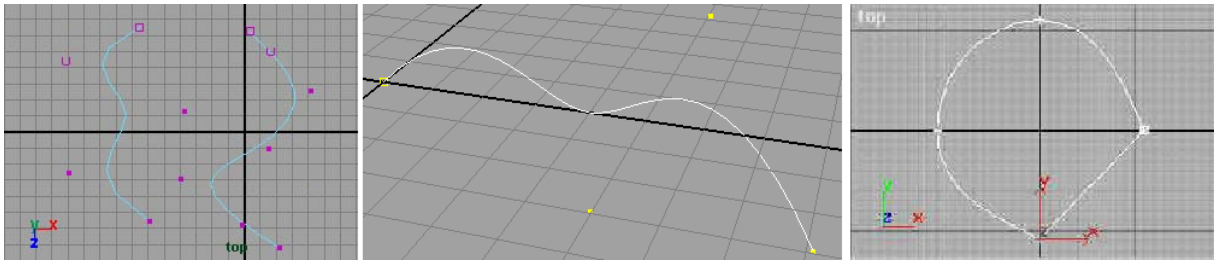


Figure 4.11: Many common 3D modeling tools, such as Maya^{TM} (*left* and *right*) and 3D Studio Max^{TM} (*middle*), provide artists with the ability to modify curve parameters.

in the vicinity of the filaments. We enable users to create a current that influences filaments to move in the desired direction of the flow by specifying curves. These *current control curves* guide the filaments on a desired path, while our simulation enforces the physical accuracy of the flow. Without resorting to introducing an unrealistic amount of *turning* contribution, these curves are designed to guide filaments with a physically realistic level of control. For example, filaments will very rarely follow current curve paths containing loops, unless the physical specifications of the system allows for the flow.

The filaments are driven against the tangents of the specified curve, much like a train is driven on its tracks. We use a customized method for tracing a filament's path according to the tangent of a specified curve. First we parameterize the B-spline control curve by its arc-length d, and assign the parameter to the filaments associated with the curve[7]. Updating a curve controlled filament is a two step procedure. Firstly, the filament is advected without the control constraint, yielding its new centroid $\bar{c}_{new}$, and coordinate frame (see section 4.2.2). Lastly, a new curve arc-length parameter, $d' = d + |\bar{c} - \bar{c}_{new}|$ is assigned to the filament and its updated average direction vector is paddled and/or turned to align with the tangent along the new curve position, $\vec{e'}_z = \tau(d')$. The new

[7]The user can associate a set of filaments with a control curve as well as giving the control curve global control privileges.

constrained filament centroid is a projection of its updated unconstrained centroid onto the plane perpendicular to $\tau(d')$, the tangent at the new curve location.

This straightforward procedure maintains the stability of the physical simulation, is inexpensive and can be performed independently on each filament. Our smoke animation and control system allows a user to animate a curve by key framing its control points. Figure 4.12 illustrates a user interface for specifying control curves in our system that is similar to curve specification tools in other graphics content creation packages. Figure 4.13 illustrates a smoke simulation with filaments following the control curve and smoke particles advected by the generated flow.

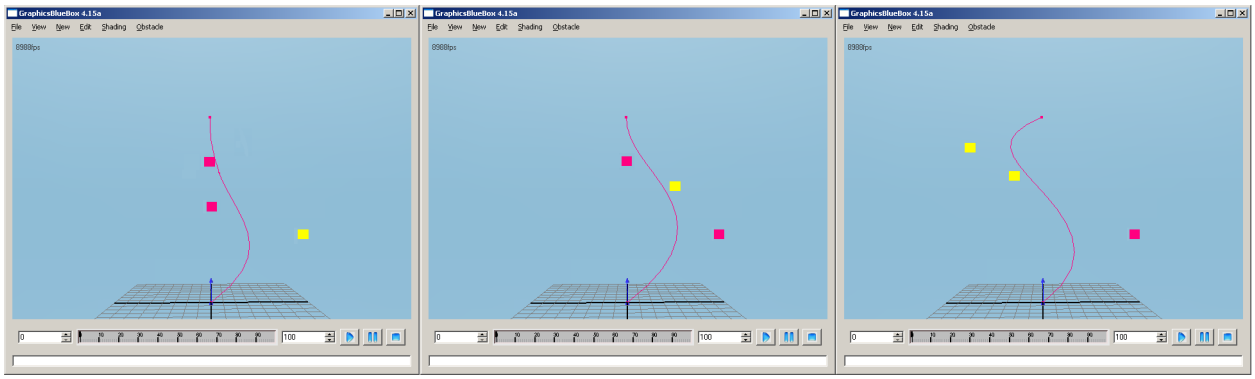


Figure 4.12: The user can specify a curve by moving one (*left* and *middle*) or more (*right*) of the curve's control points at a time.

Figure 4.13: To control the motion, the user animates a curve [5].

Current Control Attractors

If an artist simply wants to specify a intermediary flow checkpoint location, a current control curve can be used as a current control line from the smoke source to the intermediary location. However, this is a cumbersome work-around for a simple problem. Hence, we introduce another intuitive animation control called a *current control attractor* that allows the user to place a point in space as well as a spherical area of influence such that filaments within the area of influence will paddle towards the checkpoint. Once a filament reaches the checkpoint, the attraction is eliminated and the filaments are free to flow in the system's field or according to any additional active control tools in the system. Figure 4.14 illustrates the user interface visualization of an attractor. Figure 4.15 is an animation created with the same attractor configuration as in figure 4.14, illustrated with and without all system components being rendered.

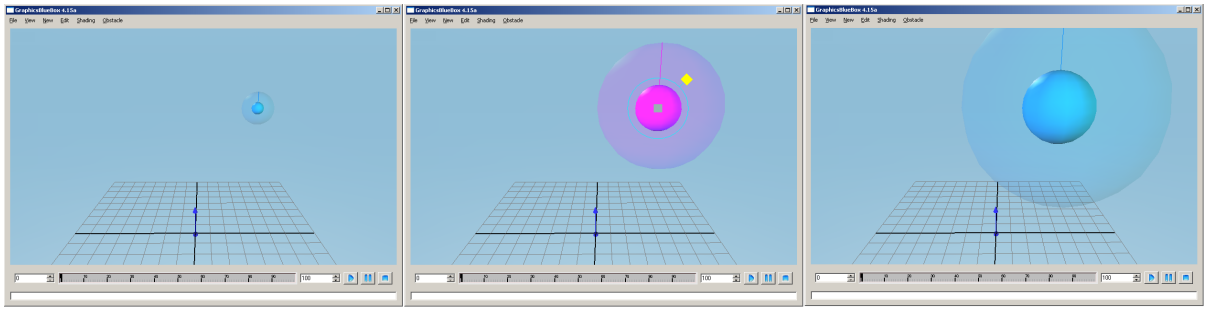


Figure 4.14: Current control attractor user interface visualization and manipulation of region of influence.

Current Tornado Effects

Another type of control we present to users is the ability to *twist* the smoke around its axis of motion yielding an up-side-down tornado effect. By modifying the Biot-Savart law, forcing it to induce velocities tangent to a filament's average direction of motion, we can achieve the desired effect. Once again, our basis decomposition allows us to

Figure 4.15: A simple animation illustrating the behaviour of smoke filaments and a current attractor, with (*top*) and without (*bottom*) UI elements illustrated. Note how filaments outside the region of influence do not deviate from their path.

determine the desired direction, yielding an equation for the twisting velocity induced by one filament on a point:

$$\vec{u}_{\text{twist}}(\bar{p}) = \int_{l\in[0,L]} \mathcal{S}K_{BS}\left(\frac{|\bar{p}-c'(l)|}{r}\right)\bar{p}-c'(l)\times\vec{e}_z \; \mathrm{d}l \quad .$$

This velocity is added to the existing velocity field of the physical system, it is not a substitute for the velocities required for regular advection. Figure 4.16 illustrates the smoke twisting effect.

4.4 Fine-level Detail with Noise

The system we have discussed so far incorporates all that is necessary to generate a realistic flow field for carrying smoke markers in a simulation. Our basis allows the user to increase the complexity of the large-scale flow behaviour, however many small-scale behaviours can not be accurately modelled without introducing small-scale filaments

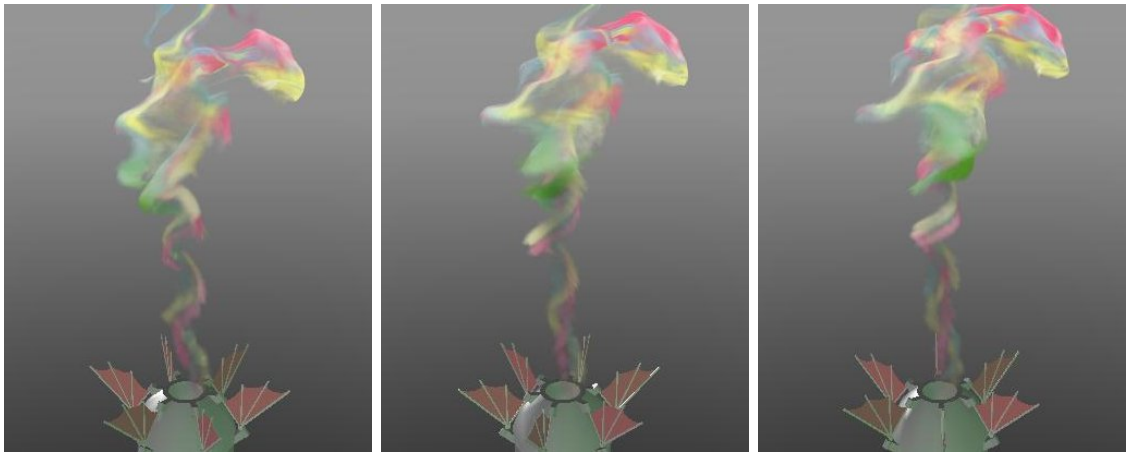


Figure 4.16: The smoke twists along its motion in a controlled manner by adding an external velocity [5].

parameterized specifically to reproduce the necessary small-scale changes in the system. Although this is feasible, we propose an alternative method of introducing this small-scale detail into our simulation based loosely on ideas previously used in the area of fluid simulation for computer graphics by Stam and Fiume in [75].

4.4.1 Previous Work

Stam and Fiume couple the deterministic behaviour of a large-scale flow simulator with a stochastic simulation of the small-scale behaviour that can be used to introduce subtle flow complexity. Assuming that the large-scale and small-scale flow fields are independent[8], we will summarize the method of Stam and Fiume used to stochastically model the small-scale flow component of the system.

Starting with a space-time distribution of random velocities $\vec{u}(\vec{x}, t)$ with component-wise averages equal to zero, the field is also assumed to be homogeneous in space and stationary in time, leading to a simplification of the cross-correlation function: it only depends on the difference between two points and the difference between their two time

[8]Which is not physically accurate.

values. This assumption allows the spatial and temporal components of the velocity field distribution to be represented in spatial and temporal frequency domains using a straightforward Fourier transformation [7]. Furthermore, the frequency domain equivalent of the cross-correlation functions, the *cross-spectral density* functions, can be obtained by simply taking the Fourier transform of the cross-correlation functions with the above formulation. Lastly, Stam and Fiume assume that the random velocity field is spatially isotropic, and coupled with the standard incompressibility constraints on the field, the work notes that the cross-spectral density functions are of the form [75]

$$\Omega_{ij}(\vec{X},\omega) = \frac{\mathcal{E}(l,\omega)}{4\pi l^4}\left(l^2\delta_{ij} - X_i X_j\right) \quad \text{for } i,j = \{1,2,3\} \quad , \tag{4.17}$$

where δ is the Kroenecker delta, $\vec{X}$ is the spatial frequency, ω is the temporal frequency, l is the length of the spatial frequency and $\mathcal{E}$ is the *energy spectrum* function of the signal. The type of small-scale contribution is controlled using different energy spectrum functions, so long as they meet the following criterion:

$$\frac{1}{2}\mathcal{A}(\vec{u^2}) = \int_0^\infty \int_{-\infty}^\infty \mathcal{E}(l,\omega) d\omega dl \quad [75], \tag{4.18}$$

Where $\mathcal{A}$ denotes the statistical averaging operator. The details for choosing an appropriate energy spectrum function can be found in [75]. Once chosen, a generalized version of Voss' inverse Fast Fourier Transform [81] is used to obtain the small-scale velocity field [75]. The contributions of [75] as they pertain to smoke particle representation and rendering will be further investigated in sections 5.1 and 5.4.

4.4.2 Our Noise Representation

Noise can be viewed as an artistic tool used to further mask the synthetic touch of an artist, and as such, we have included it in our system. To maintain the overall goal of a responsive and accurate smoke animation workflow, it is important that our detail adding

noise implementation does not interrupt or skew the effects of the control mechanisms nor reduce the performance of our system beyond an acceptable threshold. Lastly, the usability of this feature must be intuitive and simple from an artist's point of view (see chapter 6 for more details on usability).

We superpose a separate simulation responsible for generating the appropriate noise vortices over the existing smoke simulation that drives the filaments according to the system's physics and control mechanisms. To maintain animation consistency with this layered approach, we must ensure that the small-scale detail generated by the noise vortices does not interfere with the large-scale behaviour of the filaments.

The noise is generated in the spatial domain of a unit cube. A user-controllable number of vortices are placed randomly within the cube every frame. Each noise cube can be tiled with fall-off functions applied at the tiled cube faces. The area of the influence of each noise vortex is bounded by half the minimum thickness of the filaments in the system, to maintain a distinction between the amount of influence they have compared to the filaments (this is the small-scale *vs.* large-scale distinction) [5]. Figures 4.17 and 4.18 illustrate simple animations with and without noise vortices.

4.5 Summary

Although previous methods have been proposed to control the simulation of a chaotic natural phenomenon, we believe they are not sufficiently mindful of the needs of artists involved in this type of modeling nor do they provide a system that matches the workflow an artist is used to. This chapter introduced our multi-resolution basis and the control mechanisms built to leverage this basis, which allow an artist to model smoke in a manner that is intuitive, responsive and similar to other types of computer modeling tools.

Our system is designed to maintain the plausibility of the generated animation. Using a modified Biot-Savart kernel and a novel higher-order advection method, we take

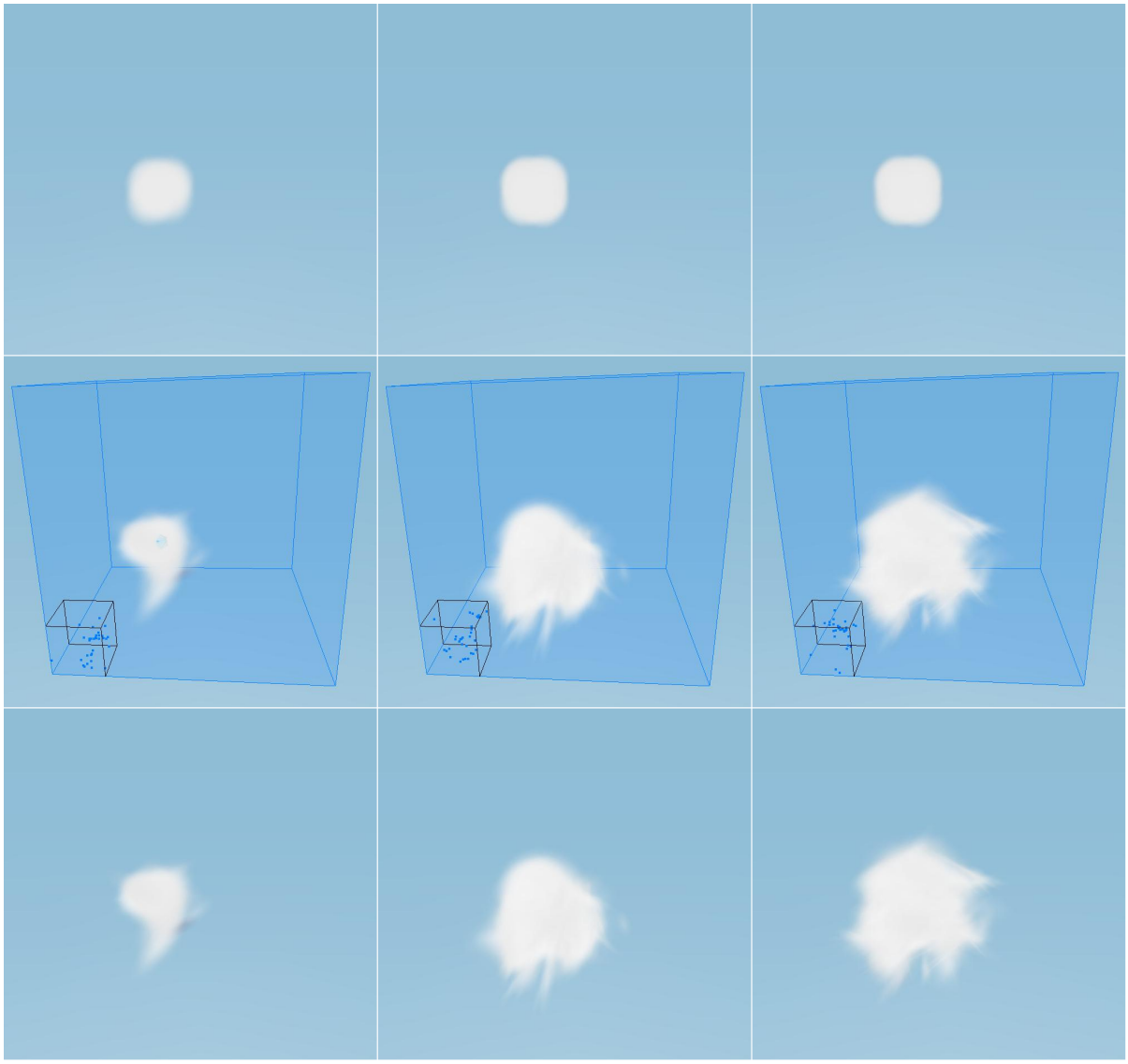


Figure 4.17: An animation illustrating the effects of noise on a smoke generator. *Top row*: the smoke generator acts without the noise's influence. *Middle row*: the smoke generator's particles are affected by the noise cube, illustrated as the base noise corner replicated throughout a larger region. *Bottom row*: the same animation as the middle row without UI components.

full advantage of our filament-based representation. Although manipulating the flow is critical to obtaining the desired smoke effects, it is the visualization of this flow with smoke particles that can captivate an audience. The behaviour of smoke in the flow as

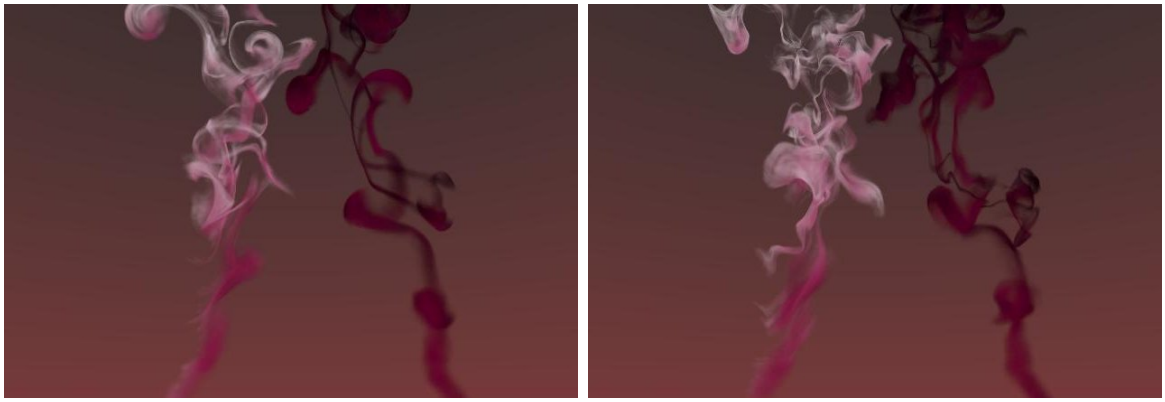


Figure 4.18: Noise disabled (*left*) and enabled (*right*). The large scale motion remains identical [5].

well as computer graphics techniques of displaying this type of participating media will be discussed in the following chapter.

Chapter 5

Smoke Particle Representation

As mentioned earlier, our physical simulator manages two separate entities: the flow carrying filaments and the smoke particles used to visualize this flow. The smoke particles are advected by the flow generated by the filaments in the system. This chapter will focus on our smoke particle model as well as the real-time visualization procedure we have implemented.

Using smoke particles for visualization allows us to isolate the advection computations in the areas of high particle density, implicitly yielding speed-ups similar to the use of adaptive spatial data structure partitioning in grid-based simulators. Furthermore, particle systems are straightforward to integrate into real-time and offline rendering algorithms. Each particle of smoke in our system has various parameters, including diffuse and shadowing colors[1], size and density. We will elaborate on some these properties in section 5.2.

The properties of smoke particles as well as smoke generators can be tuned by an artist to achieve the desired effect; section 6.3 illustrates the widgets and forms used to easily control these parameters. We will first examine previous work on the representation and visualization of smoke particles and densities.

[1]The physical process underlying the perceived color of different types of smoke is very complicated to simulate, thus we simply allow an artist the ability to assign colors manually.

5.1 Previous Work

In section 4.4 we introduced the work of Stam and Fiume that separated the small and large scale detail of a flow field. They also separate the flow characteristics from the actual material characteristics of their simulator: instead of representing individual smoke particles, Stam and Fiume consider the effects of their flow fields on smoke densities[2] [75].

Stam and Fiume propose a temporal density distribution composed of a sum of weighted Gaussian distributions with standard deviations significantly smaller than the finest scale of detail of the system's flow field. These assumptions allow for an analytic formulation of the evolution of the density distribution due to the effects of the flow field on the individual Gaussian components. They also propose a modification to the standard ray tracing algorithm to handle the effects of participating media in a scene. The intensity of light visible through smoke is determined with a front-to-back attenuation algorithm. Since smoke, unlike typical geometrical objects in a virtual scene, does not have a fixed boundary to intersect a ray, intersection of every ray is tested against the individual Gaussian density distributions. Furthermore, the ray-intersection test does not only maintain the closest intersection points: each ray is subdivided into intervals, using the entry and along the intervals according to the transparency value of each blob at the relevant positions in the blob [75]. Figure 5.1 below illustrates the results obtained with this rendering algorithm.

Fedkiw *et al.* present two algorithms for visualizing the smoke densities stored in a grid [19]. The first is a hardware-accelerated rendering algorithm based on the *Stable Fluids* rendering algorithm [74]. A Bresenham line drawing voxel traversal algorithm [42] is used to determine the amount of light entering at each voxel in the grid, and appropriate voxel transparencies are roughly approximated with an exponential drop-off dependent on the smoke's extinction coefficient and the voxel width. Once the approximate radiance

[2] Assuming that these densities do not affect the flow field and are governed by the advection-diffusion process.

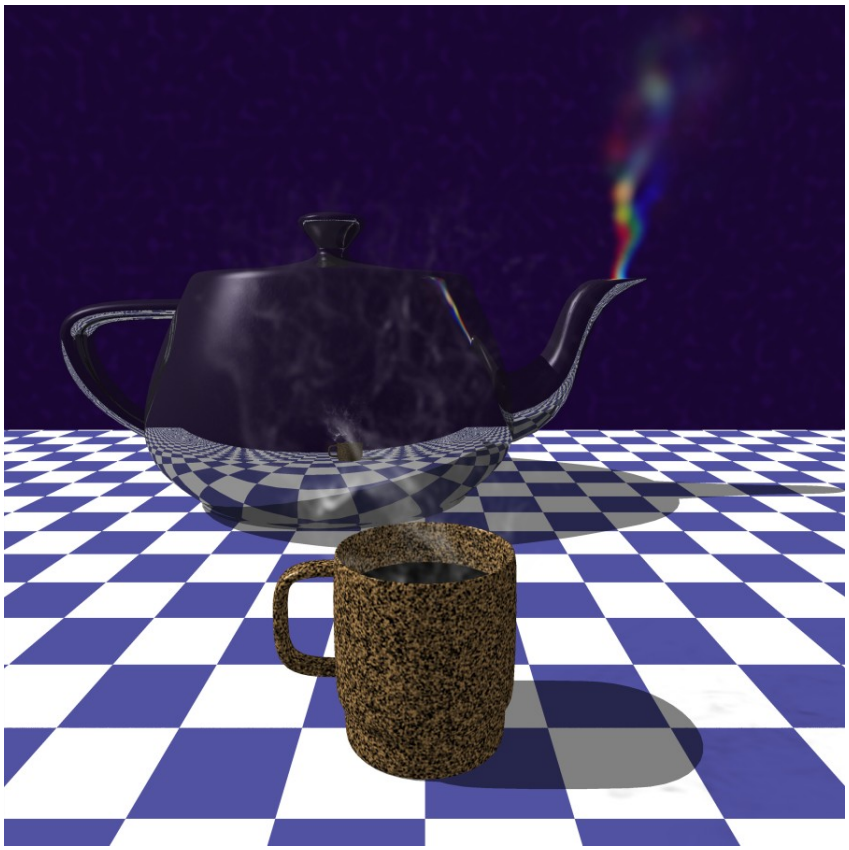


Figure 5.1: The results of Stam and Fiume's offline rendering algorithm [75].

is set using this information at each voxel, grids are sliced along the view direction and each slice is rendered as a quad in front-to-back order, alpha-blending the transparency along the way [19].

Fedkiw *et al.* propose a high-quality offline renderer to generate images from the simulation with a more regimented analysis of the scattering mechanisms involved in a participating medium. A modified two-pass photon-mapping algorithm is used for this purpose. During the first pass, indirect illumination photons are stored in a volume photon-map [37]. The second pass implements a novel *forward marching* ray tracing algorithm gathering radiance of the form

$$L_n(p_n, \vec{\omega}_o) = L_{n-1}(p_{n-1}, \vec{\omega}_o) + \exp\left(-\tau(p_n)\right)\ \Delta p_n\ L_{inscattered}(\epsilon, \vec{\omega}_o) \quad [19], \tag{5.1}$$

where L_n is the radiance at p_n, the n^{th} scattered intersection point of the ray, $\vec{\omega}_o$ is the outgoing view direction, $\tau(p) = \int_{p_0}^{p_n} \sigma\ dp$ is the *optical depth* with extension coefficient σ, $L_{inscattered}$ is the inscattered radiance at a randomly chosen location ϵ along the current ray segment, and Δp_n is the step size of the ray segment. The inscattered radiance is defined, in terms of a single scattering term, L_s, and a multiple scattering term, L_m as

$$L_{inscattered}(p, \vec{\omega}_o) = \alpha\ \sigma(p) \int_{4\pi} \left(L_s(p, \vec{\omega}) + L_m(p, \vec{\omega})\right)\ \beta(\vec{\omega})\ d\vec{\omega} \quad [19], \tag{5.2}$$

where α is the albedo[3] of the medium and β is the local scattering profile of the light. Typically, β is derived from the *Henyey-Greenstein* phase function first proposed in the graphics literature by Hanrahan *et al.* [34] who also note that the choice of scattering profile function can be tuned.

The single scattering radiance term is computed with a ray marching algorithm that travels along intervals of a ray and determines whether the ray's direction should be altered, and the multiple scattering radiance is reconstructed from the volume photon-map using the following gathering kernel:

$$L_m(p, \vec{\omega}) = \frac{1}{\sigma} \sum_{1}^{N} \frac{P(\vec{\omega'})\ \beta(\vec{\omega'})}{\frac{4}{3}\pi r^3} \quad [19], \tag{5.3}$$

where N is the number of photons used during reconstruction, r is the radius of the sphere enclosing the nearest N photons to position $\overline{p}$ and P is the power of each photon. Figure 5.2, as reproduced from figure 3.3, illustrates the results of the physically-based photon-map renderer. We will briefly overview some visualization techniques used in Lagrangian simulations of water and smoke.

[3]Which can be thought of as either the probability of scattering, or the translucency of the smoke.

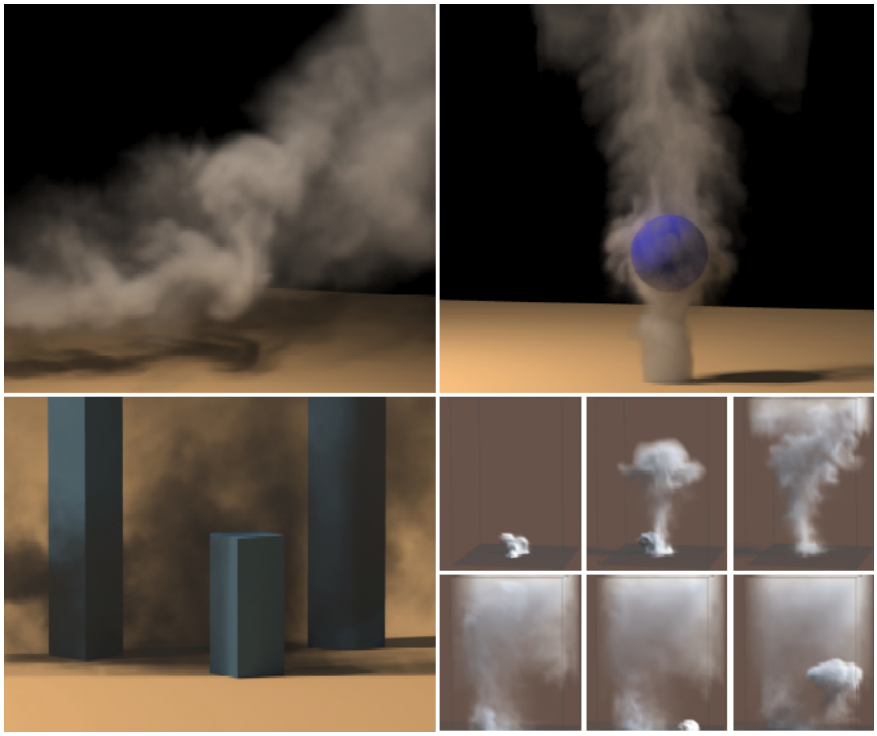


Figure 5.2: A physically-based photon-map rendering of smoke [19].

The work of Müller *et al.* introduced in section 3.1.2 used density distributions (defined with weighted kernels) to represent particle regions. Müller *et al.* define a *color field*, in a fashion similar to the density field definition, as:

$$C(\vec{x}) = \sum_i \frac{m_i}{\rho_i} W(\vec{x} - \overrightarrow{x_i}, h) \quad [55]. \tag{5.4}$$

The normal and curvature of the surface field are the first and second moments of the color field. Müller *et al.* define surface particles as density centroids that satisfy the following condition:

$$\nabla C(\vec{x}) > l \ ,$$

where l is a user defined threshold parameter. With this information, a set of unconnected surface points can be determined for use with a point-splatting [84] algorithm to visualize

a plausible surface with interactive results. Alternatively, Müller *et al.* triangulate a iso-surface of the color field using the marching-cubes algorithm. This procedure is time consuming, and thus cannot be used for the interactive rendering of the water surface, but the results are smoother and more realistic than those of the surface splatting technique. Figure 3.5 illustrates the results of the marching cubes visualization algorithm.

As with the work of Fedkiw *et al.*, Park and Kim describe two different rendering alternatives for their vortex-based particle simulator. The first is an offline renderer based on the *Stable Fluids* work: densities are stored on an overlaid grid and these densities are advected in a Eulerian fashion using velocities defined by the Lagrangian vortex particles. Visualization of the density field is performed in a similar fashion to [74] and [19]. However, the use of a secondary rendering grid side-steps the potential benefits of their Lagrangian formulation. Alternatively, Park and Kim describe how neutral particles can be introduced into the system for visualization purposes. These particles are advected with the velocity generated by the vorticity field, and a standard texture billboarding algorithm is used to visualize the particles. The latter rendering alternative's quality depends substantially on the distribution of visualization particles. If too few are used, the resulting smoke may look patchy; if the distribution of the visualization particles is not set to sample the field variation adequately, then the resulting smoke may not represent the simulated flow accordingly. Figure 5.3 illustrates the second, real-time billboarding results of Park and Kim.

With this background we shall proceed with an outline of our smoke particle representation, advection and adaptive particle splitting.

5.2 Our Smoke Particles

In contrast to the offline method proposed by Park and Kim, the rendering procedures outlined from Fedkiw *et al.* and *Stable Fluids*, we do not introduce a separate density

Figure 5.3: Park and Kim's real-time billboard rendering algorithm results [58].

grid for rendering purposes; instead, we propose the use of *adaptive smoke particles* coupled with a real-time rendering algorithm. This algorithm, covered in section 5.4, is computationally cheap in both time and memory and produces surprisingly convincing results compared to previous offline and real-time smoke renderers.

Using particles instead of grid-based densities typically accelerates the rendering procedure, since processing time can be focused only on the regions of interest (the particle locations), and takes advantage of the benefits of our Lagrangian simulation. Our system supports two types of particles: standard and adaptive. Standard particles are used to simulate large scale phenomena, such as the cloud of smoke from a volcano eruption. These particles have larger opacities and do not respond to small scale perturbations. Adaptive particles are used to represent smaller scale phenomena, such as the wispy smoke from an incense stick. These particles bend and twist as well as split adaptively if they are deformed past a threshold during the simulation [5]. Both types of particles have the same underlying representation; it is the various particle settings (such as size, density and splitting) that allow us to distinguish between what type of particle we are currently using. Figure 5.4 below illustrates how different particle parameter settings yield drastically different results for the same system layout.

The deformation mechanism and the splitting procedure used with adaptive particles

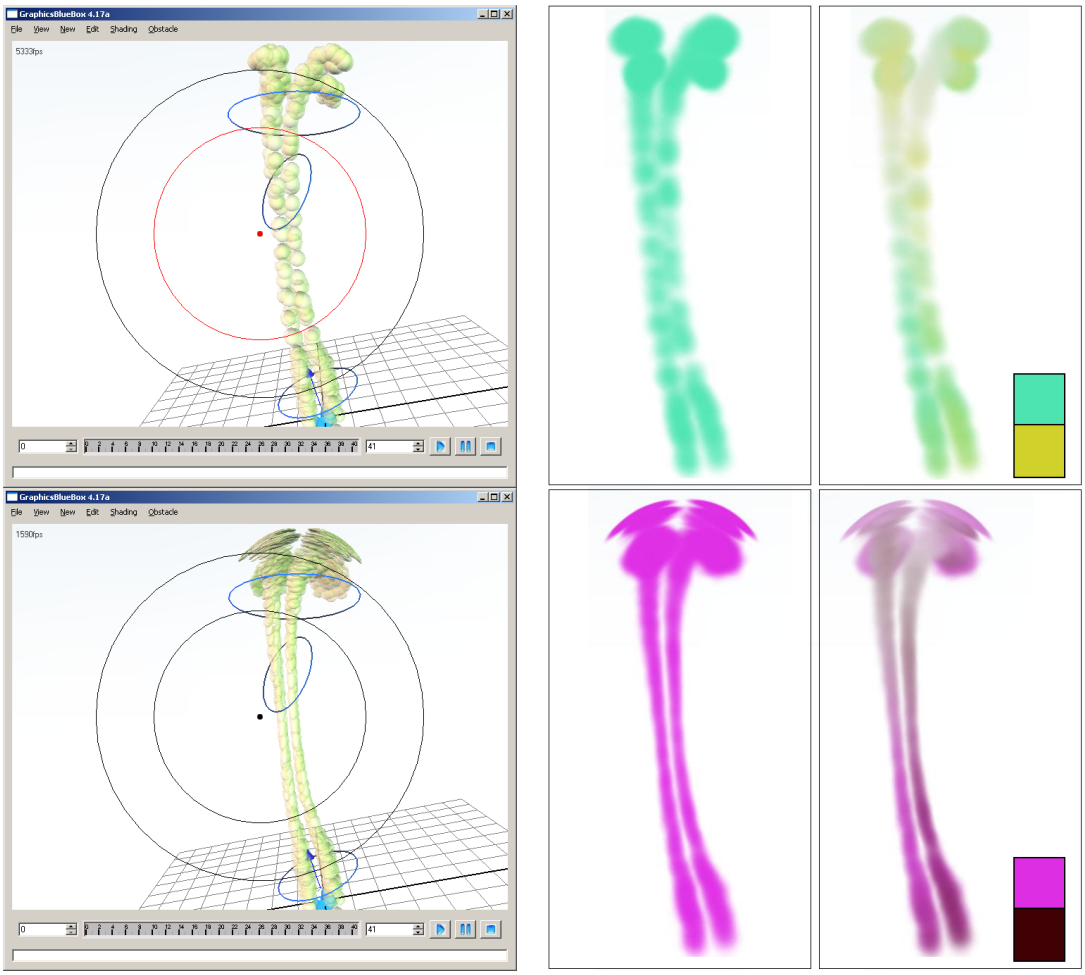


Figure 5.4: Modifying smoke particle parameters. *Top row*: Particles with adaptive splitting disabled. *Left column*: Our animation package illustrating the design procedure for the animation sequence, with smoke particles rendered as un-shaded ellipsoids. *Middle column*: Shaded particles without our self-shadowing rendering algorithm. *Right column*: Shaded particles with our self-shadowing algorithm; the *inset* identifies the diffuse (*top*) and shadowed (*bottom*) color settings that our self-shadowing algorithm uses to determine the final pixel colors.

will be discussed, followed by the advection of particles and our novel particle rendering algorithm.

5.2.1 Adaptive Particle Stretching and Splitting

Our smoke particles stretch according to the advection forces applied to them during the simulation. If particle splitting is enabled, the smoke particles split to avoid undersampling due to excessive stretching. The method we use to determine the amount of stretching a particle undergoes during advection is based on principles of space deformations. If we abstract the notion of a smoke particle to that of a volume of matter as well as abstract the advection forces to a deformation function, we can analyze the stretching of a particle in terms of the deformation of a volume. Figure 5.5 illustrates the deformation of a volume due to a function $\mathcal{F} : \mathbf{R}^3 \longmapsto \mathbf{R}^3$.

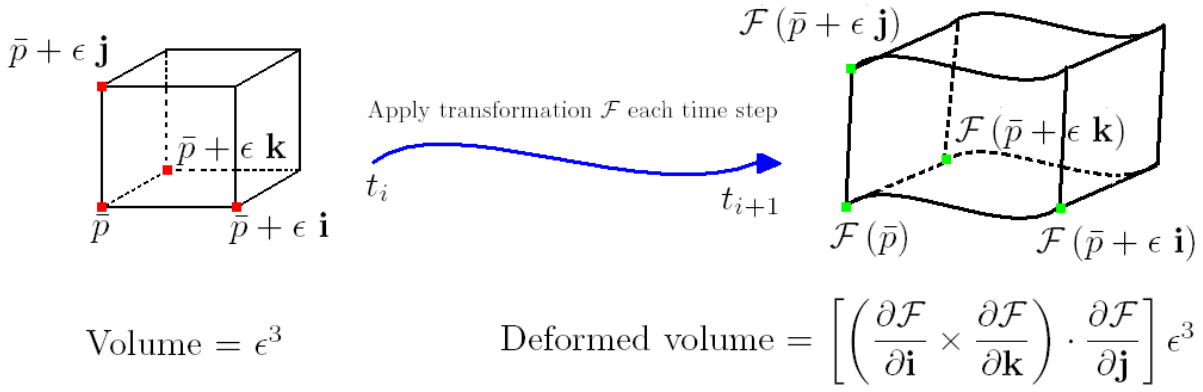


Figure 5.5: Volume calculations after a deformation function, $\mathcal{F}$, is applied on the original shape ($left$). Original and transformed points are identified in red and green.

Let us define the *Jacobian* of the deformation as $\mathbf{J} = \left(\frac{\partial \mathcal{F}}{\partial \mathbf{i}} \ \frac{\partial \mathcal{F}}{\partial \mathbf{j}} \ \frac{\partial \mathcal{F}}{\partial \mathbf{k}}\right)$. The determinant of the Jacobian is equal to the ratio of the deformed volume to the undeformed volume. For an incompressible fluid, this determinant must be unity. In the case of our simulator, the partial derivatives of the deformation are discretely approximated every time step. For example, $\frac{\partial \mathcal{F}}{\partial \mathbf{i}} = \frac{\mathcal{F}(\bar{p} + \epsilon\ \mathbf{i}) - \mathcal{F}(\bar{p})}{\epsilon}$. Some fluid simulators perform many simulation steps prior to displaying a rendered frame[4], however our simulator performs one

[4]This is typically done to allow for a reduced time step size for each simulation step in hopes of

simulation step per rendering step. If we were to perform multiple simulation steps prior to each rendering step, we could use these simulation results to better approximate the partial derivatives required to obtain the Jacobian of the deformation applied to each smoke particle. To re-align ourselves from shape modelling terminology back to physical dynamics' terminology, the Jacobian matrix is equivalent to the *Displacement Gradient Tensor* [2].

We initialize every smoke particle as a spherical volume. During simulation, the stretching effect is accumulated every time step and applied to each spherical smoke particle, and they are approximated by non-uniform ellipsoids. The parameters of the updated ellipsoids of each particle are obtained efficiently using the eigenvectors and eigenvalues of the *Metric Tensor*, $\mathbf{M}$, derived from the *Displacement Gradient Tensor*:

$$\mathbf{M} = \mathbf{J}\mathbf{J}^T \quad [6,\ 5]. \tag{5.5}$$

In detail, each smoke particle is initialized as a sphere with eigenvectors $(\vec{e}_x, \vec{e}_y, \vec{e}_z)$ and eigenvalues (v_1, v_2, v_3) which define the principal axes (which initially coincide with the world coordinate axes $(\mathbf{i}, \mathbf{j}, \mathbf{k})$) and radii along those axes of the particle. This data is represented in a matrix of covariance C. The covariance matrix is updated after deformation using the *Metric Tensor* as follows:

$$C_{new} = \mathbf{J} \cdot C \cdot \mathbf{J}^T \quad [1,\ 4,\ 6]. \tag{5.6}$$

Therefore, the updated eigenvectors and eigenvalues of C_{new} define the updated ellipsoidal shape of a deformed smoke particle. Figure 5.6 illustrates the stretching of particles from spheres to ellipsoids during simulation.

If the amount of stretch in any one of the three principal directions exceeds an absolute threshold, the particle is split along the over-stretched axis, say $\vec{e}_x$, into two particles with

minimizing numerical dissipation effects introduced by increased time step interval lengths.

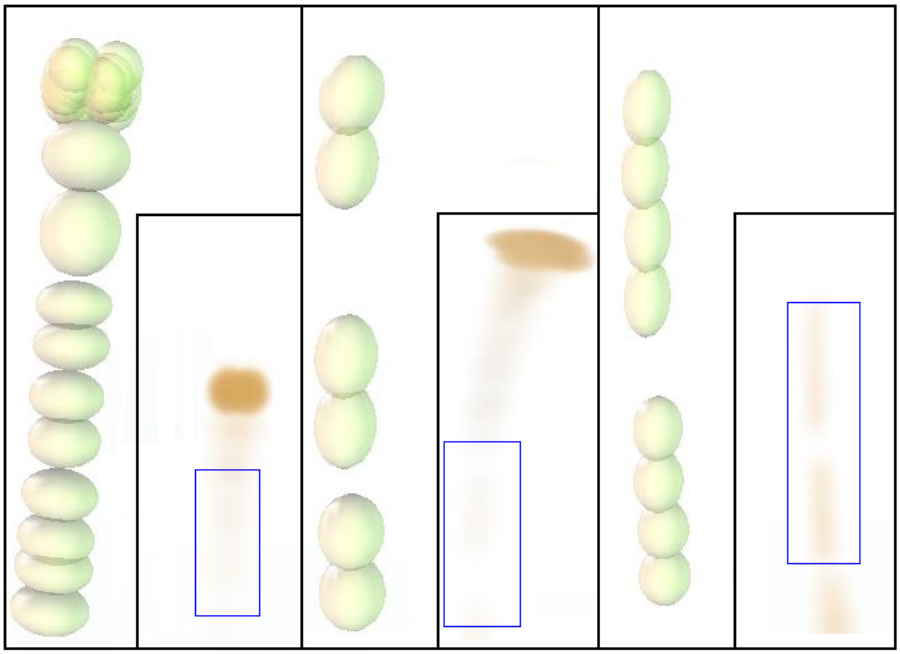


Figure 5.6: Three close-ups of regions (*blue rectangles*) of an animation sequence illustrating the stretching of smoke particle ellipsoids. Close-ups are rendered as flat ellipsoidal particles without shading.

new centers $\{p_1, p_2\} = p_{old} \pm \frac{\zeta}{2}\vec{e}_x$, where $\zeta = \sqrt{2v_1}$ is the amount of stretching along the axis [6, 1]. Figure 5.7 illustrates the difference between an animation with and without particle splitting enabled. The frequency of smoke particle output is intentionally defined to be low to emphasize the effects of splitting. The sampling and smoothing benefits of particle splitting are more evident in a scene with a sparse distribution of particles.

5.3 Smoke Particle Advection

Smoke particles are advected in the flow as passive markers using the velocity field derived from the vorticity field of the system. As with filaments, we advect smoke particles using the higher-order advection procedure described in section 4.2.3 and our modified Biot-Savart kernel. The smoke particle positions are updated with equation 4.13, which also

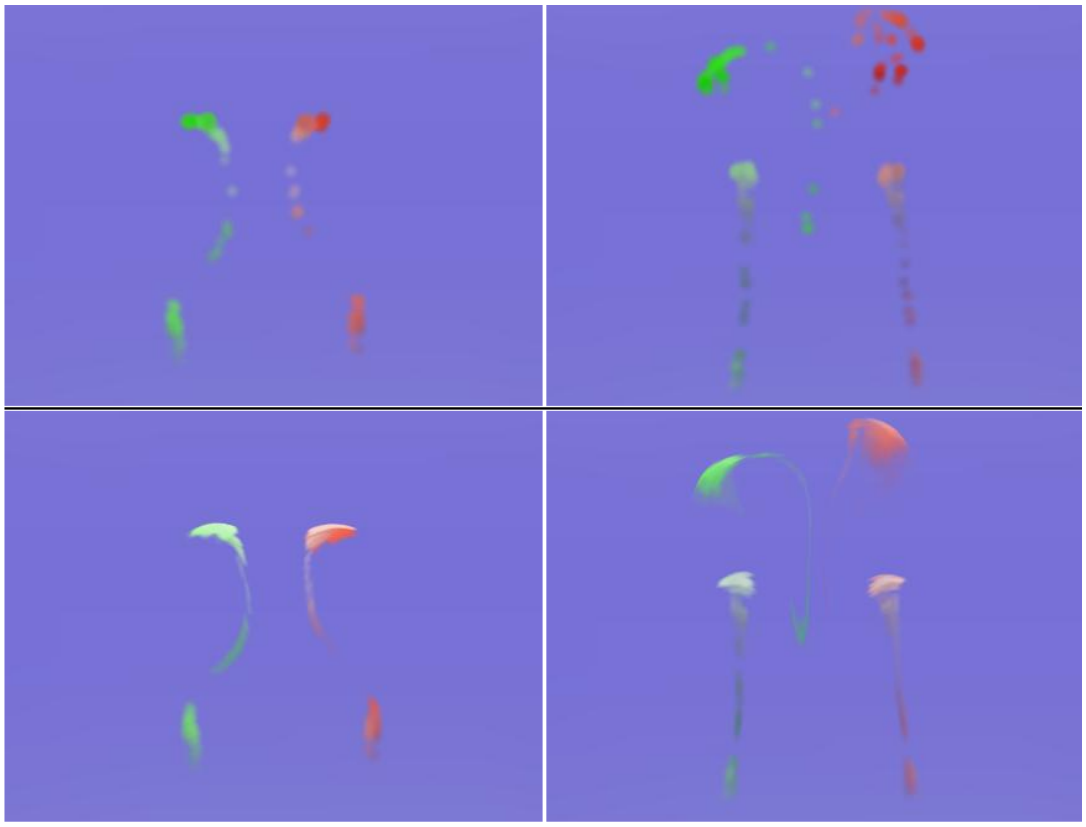


Figure 5.7: Identical frames of an animation with two smoke swirls with (*bottom row*) adaptive smoke particle splitting and without (*top row*).

provides us with the the base from which we can perform the covariance update for particle stretching and splitting described in the previous section.

5.4 Particle Shading Model

We have discussed previous techniques used for smoke rendering. These techniques apply different illumination models (some physically-based and some completely ad-hoc) to obtain the final lighting effects due to smoke. We proposed a simple yet compelling illumination model for rendering our adaptive smoke particles with real-time performance similar to the technique originally presented by Angelidis and Neyret [4].

In order to enhance the visual quality of our results, we simulate a self-shadowing

process by sorting particles in front-to-back order according to the viewpoint and collecting particle opacities during rendering (this feature can be toggled for a slight performance enhancement, if necessary). Once sorted, we project each ellipsoidal particle onto a two-dimensional viewer-aligned billboard and calculate the shading according to a density gradient defined over each particle.

Projecting a particle onto a viewer-aligned billboard requires obtaining orthogonal view plane axes, $\mathbf{x}$ and $\mathbf{y}$ and then simply performing the following operation to obtain a covariance matrix describing a two-dimensional image-plane projected ellipse:

$$C_{view} = (\mathbf{x}\ \mathbf{y})^T \cdot C \cdot (\mathbf{x}\ \mathbf{y}) \quad [1]. \tag{5.7}$$

User adjustable parameters, such as color, are combined with our diffuse shading model to obtain the color gradients blending throughout each projected billboard, resulting in a smoothly shaded smoke particle without any perceivable boundaries. This visual cue is often sufficient, in the perceptual sense, to convince a viewer that our smoke particles are not composed of geometric primitives, as with other computer rendered objects. Unlike many two-dimensional billboarding effects, we maintain a three-dimensional visual representation of each smoke particle by assigning the gradient-adjusted color using a three-dimensional shading model. The most important piece of information required for shading our particles is the normal to each particle. Thin smoke often exhibits surface like visual behaviour despite our previous statement regarding the disassociation between smoke particles and geometric surfaces, thus we emphasize the importance of obtaining a normal for shading.

The gradient of the density of a ellipsoidal smoke particle near the boundaries of the particle define the normal at the surface [5]. We can obtain the normal analytically given the viewer's eye position:

$$\vec{n} = \left(\mathbf{J}\mathbf{J}^T\right)^{-1}\mathbf{e} \quad [5,\ 2], \tag{5.8}$$

where $\mathbf{e}$ is the viewer's eye position in the same co-ordinate frame as the normal (typically the world co-ordinate frame). Furthermore, we can incorporate the shading of smoke using the normal information with our hybrid image-based billboard rendering algorithm. If we already have normal information, we could determine the deformed normal of an ellipsoid after it has been advected by one time step with the following equation:

$$\vec{n}_{deformed} = \frac{\partial \mathcal{F}(\bar{p})}{\partial \mathbf{x}} \times \frac{\partial \mathcal{F}(\bar{p})}{\partial \mathbf{y}} \tag{5.9}$$

$$= \mathbf{J}^{-T} \cdot \vec{n} \quad [2], \tag{5.10}$$

however we can only use this formulation if the viewer position does not change. Since equation 5.8 is computationally efficient, we simply recompute the normals of the particles during each rendering pass. Figure 5.8 illustrates a profile of the viewer's eye position in relation to a single smoke particle, as well as the particles normal for one particular viewing direction.

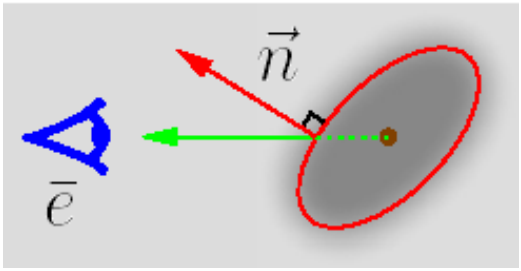


Figure 5.8: The normal vector $\vec{n}$ is derived according to the local gradient of density of a smoke particle. $\bar{e}$ is the viewer's eye location [5].

Figure 5.9 (as well as the second and third columns of figure 5.4) illustrates the difference between a frame rendered with and without our view-dependent depth-sorting

algorithm. With the depth-sorting and opacity collection algorithm enabled, the smoke seems to shadow itself realistically. This effect, although substantial, incurs a very small performance price. In fact, for most of the animation sequences generated for this thesis, the self-shadowing rendering algorithm uses up about one percent of the total CPU processing power.

Figure 5.9: Self-shadowing disabled ($left$) and enabled ($right$) [5].

We justify using a real-time *physically-motivated* rendering algorithm instead of an offline *physically-based*[5] rendering algorithm with our original motivations of empowering an artist with a workflow solution that does not constrain the artist within the computational confines of an overly-complicated system.

5.5 Summary

Leveraging the benefits of a Lagrangian simulation, we proposed and outlined our particle representation in this chapter. Various particle parameters can be adjusted by the user of our system. For simulation purposes, adaptive splitting of smoke particles is the most significant parameter which allows an artist to represent thin, wispy smoke more

[5]The terms *physically-based* and *physically-motivated* are sometimes confused in computer graphics literature. We refer to methods that aim to strictly enforce physical laws, such as offline volumetric path rendering algorithms, as *physically-based*, and methods that model themselves after approximations based on physical laws as *physically-motivated*.

realistically. For rendering purposes, our physically-motivated self-shadowing shading model is the most significant effect that an artist can control.

The compact mathematical representation of each particle is used to track the deformation of the smoke as it is advected, and this representation is used unchanged in our rendering module. Furthermore, we can calculate the strain and other useful physical properties using the representation described in this chapter.

Finally, although our system uses a high-performance real-time shading algorithm in order to create a seamless user experience, the software architecture of our system (see chapter 6) allows for the addition of a more accurate, physically-based offline rendering algorithm. Our rendering algorithm does sometimes yield results that are visibly unrealistic, such as when the density of particles is too coarse and the visual space is poorly sampled. This can occur when the smoke particle emission rate is too slow. Enabling particle splitting nearly always increases the particle density to a point where our results become realistic. This sampling issue is present in figure 5.7.

Chapter 6

Implementation

6.1 Graphics Blue Box Architecture

Our smoke simulation system is implemented as an extension, or software plug-in, on top of an existing open-source computer graphics software framework called the *Graphics Blue Box* (GBB). This open-source platform is developed by Alexis Angelidis with the goal of balancing a lightweight architecture with a robust and extensible framework similar to Maya's API. Also, the GBB implements a standard set of graphical data structures, such as curves, bounding boxes, vector/matrix operations, frustum culling and scene graphs.

The architectural framework of the GBB is based on the exchange of data through connected nodes. We will briefly outline the behaviour of data exchange under the GBB architecture in the following section.

6.1.1 GBB Data Management

Much like the underlying development platform of Maya, the GBB introduces the concept of connected nodes as a means of encapsulating data and exchanging information between the separate components of a functional software add-on. The GBB uses *nodes* and *plugs*

at its foundation for data exchange [2].

Nodes

In the GBB architecture, nodes are designed as high-level data encapsulator and transfer mechanisms, much like Maya's nodes. Unlike Maya's node structure, only one line of code is required to add an attribute (of arbitrary type, in the programming sense) to a node in the GBB. Nodes may also contain sub-nodes, and a developer can designate input and output node attributes. Connectivity of attributes across separate nodes is defined using *plugs*.

Each node must have an `initialize` function and a `compute` function. Moreover, each node must be registered with the GBB's node management system. Internal node data attributes as well as the connectivity information between data attributes inside the node and external nodes are defined within the `initialize` method. The `compute` method's sole parameter is a generic `Plug` data type. This method is responsible for generating output for data attributes that are connected to other attributes, whether internal to the node or from external nodes. The `compute` method compares the requested output plug with the potential output data attributes of the node and, upon determining the appropriate match, computes the required information and forwards it to the requesting plug object.

Registering a node object adds it to the scene graph management system built into the GBB. The reserved `Scene` node name is automatically initialized as the root of the scene graph during the start-up procedure of the GBB. The GBB's command line functionality allows a user to manually access the property window, listing all the UI-bound input and output data attributes of a node, by explicitly entering the full scene graph path of the desired object. Alternatively, if the object of interest is a rendered object that is visible from the current view orientation, clicking the object and pressing **CTRL + A** will also bring up the property window. Figure 6.1 is a screen capture of a user manually

requesting the property window of the default camera object of the scene by using a command prompt query with the object's full path (*red*).

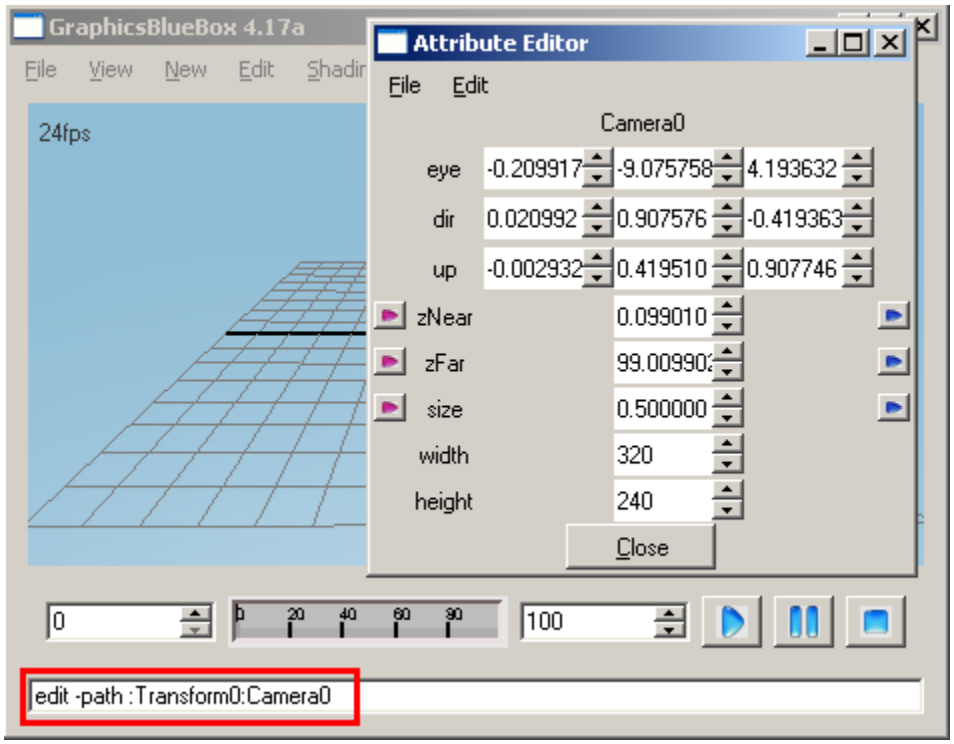


Figure 6.1: Using the GBB's command prompt to access the property window of the `Camera0` object.

The GBB UI automatically uses the appropriate number and type of windowing interaction components to display the attributes of a node in its property window. For example, in figure 6.1 above, three adjacent scrollable input boxes are used to represent vector-type attributes, such as `eye`, `dir` and `up`, which define the orientation properties of the camera object. Not visible in this example, checkmark input boxes are used to represent boolean node data types. The developer may also define upper and lower limits for integer and float data types. These limits are enforced by the UI system.

Plugs

Each data attribute has a *plug* object associated with it. Apart from allowing the developer to associate directional connectivity between attributes [1], the plug mechanism also handles the maintenance of data values and the proper propagation of values throughout the underlying system. The pink and blue arrow buttons next to attributes in an object's property window (see figure 6.1) allow a user to request a curve editor window for defining time varying data values. Figure 6.2 illustrates this functionality.

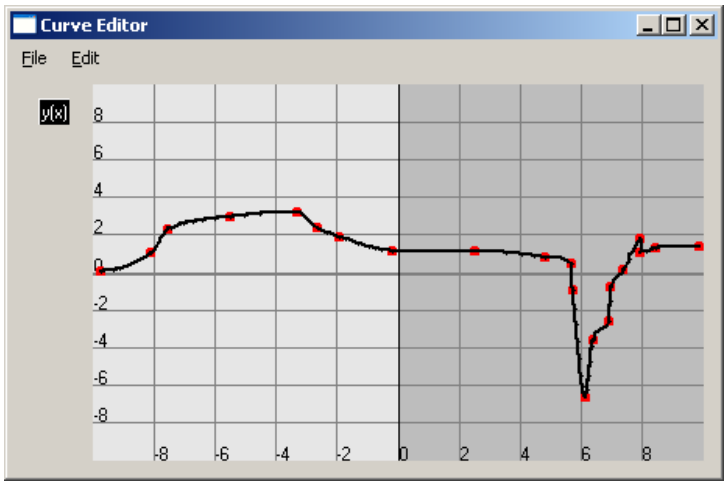


Figure 6.2: Curves can be defined to vary the value of a data attribute over time.

These time varying data values are handled appropriately by the plug mechanism. The procedure of data attribute propagation will be discussed in section 6.4.1. Data attribute values may also be key-framed by the user. The plug architecture manages the key-framing of data attribute values, as well as a interpolation of the values between key-frames using a simple quadratic interpolator. As with *nodes*, plugs may also consist of sub-plugs. The UI features of the GBB, such as the property window and curve editor, are specifically designed with computer artists in mind and thus maintains the objective of an artist-friendly work environment for our smoke animation package.

[1] One input plug may be associated with many output plugs, and vice-versa.

6.1.2 User Interface

The previous section illustrated the main UI construction of the property and the curve editor windows. We will briefly overview the other main UI features of the GBB. The most used UI tool of the GBB is the aptly named *gizmo* illustrated in figure 6.3. This widget allows a user to select, scale, rotate and translate any visible object. Furthermore, once selected, an object's position and orientation can by key-framed and its property window can be opened. This type of on-screen object manipulation is common to many standard computer animation tools.

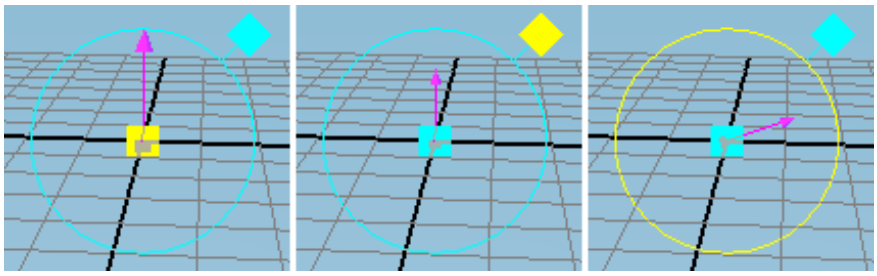


Figure 6.3: A gizmo selecting, scaling and rotating a current generator object.

Another useful GBB feature is the ability to key-frame *any* presented data attribute. As with Maya, key-frames are tracked on a per-object basis and setting/deleting key-frames is a key-stroke (with optional menu item) operation. Figure 6.4 illustrates the key-framing and interpolation of various orientation properties of a current generator object.

6.2 Simulation Flow

We refer to [5] for a simple five step breakdown of each time step of our animation engine:

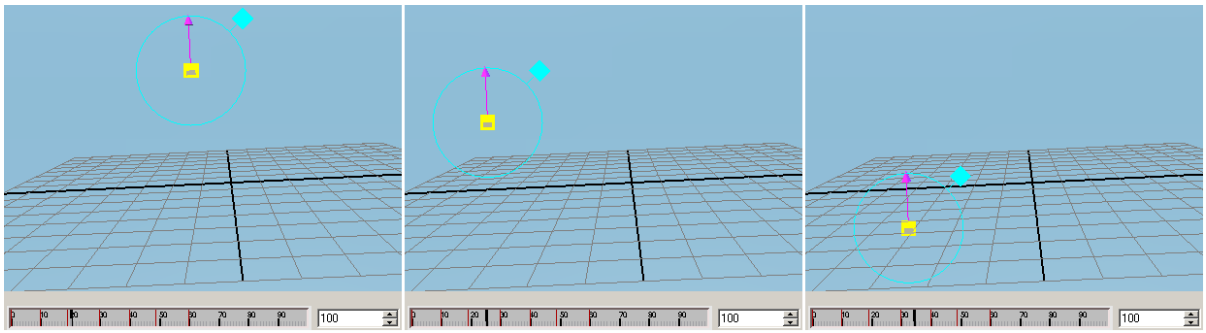
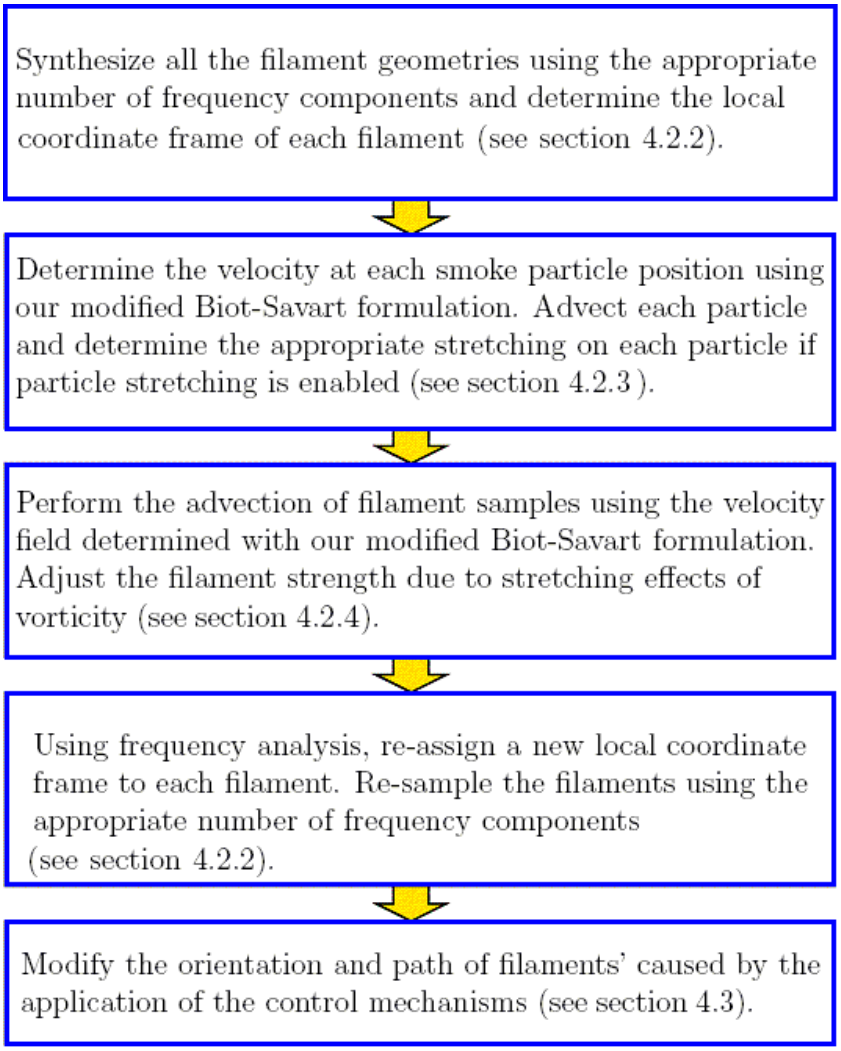


Figure 6.4: Key-framing and interpolation of current generator orientation.



6.3 Smoke Simulation and Control Software Components

As mentioned at the beginning of this chapter, our smoke animation system is built on top of the GBB architecture. Our smoke animation extension is implemented as a set of

interacting GBB nodes. Of this set, will discuss three major node types, two minor node types and a brief summary of some control node types. Instead of overviewing the details of these nodes from a developer's perspective, we chose to summarize the key attributes and properties of these nodes from a user perspective. Most of the major, minor and control nodes implement a `render` routine, allowing a user to visualize their behaviour.

The three major node types we will overview are the `CurrentGenerator`, `SmokeGenerator` and `SmokeParticles` node types. The two minor node types we will overview are the `VortexRing` and the `VortexSet` nodes. The control node types we will investigate are the `CurrentAttractor` and `CurrentTurbulence` nodes. A description of these nodes, as well their key parameters and screenshots will be provided.

6.3.1 `CurrentGenerator` Node

The `CurrentGenerator` node is responsible for generating vorticity carrying filaments. Each filament is created as a nested node of type `VortexRing`. This node type will be discussed in section 6.3.4. The `CurrentGenerator` creates a new filament on the appropriate time steps and enqueues the filament into the appropriate processing bin according to whether or not the current generator object has a control entity, such as a current control curve, associated to it. Each generated filament has its initial local coordinate frame calculated by the current generator object. Figure 6.5 is a screenshot of a current generator's visual widget as well as its property window. Table 6.1 describes the key attributes of the current generator.

6.3.2 `SmokeGenerator` Node

The `SmokeGenerator` node's function is to introduce smoke particles into the physical system. The initial distribution of smoke particles can be contained in a cubic region (see figure 6.6, top right) or a region defined by a torus (see figure 6.6, bottom right). The properties of the generated particles can be modified using the `SmokeGenerator`

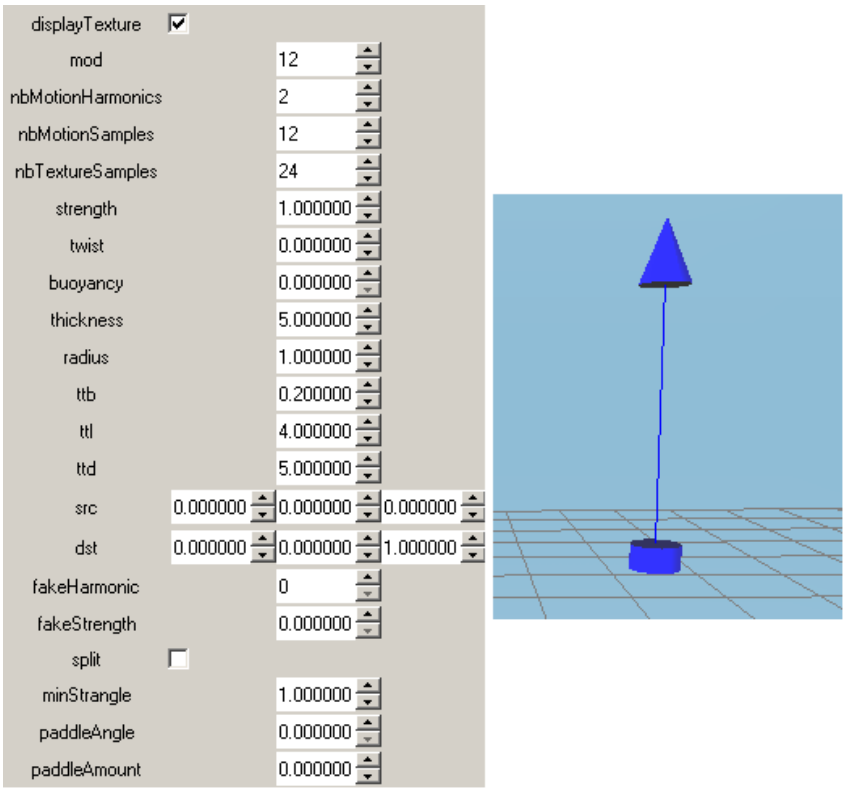


Figure 6.5: A current generator's property window and visualization widget.

attributes coupled with the attributes of the `SmokeParticles` node (see below). Table 6.2 describes the key attributes of the smoke generator.

6.3.3 `SmokeParticles` Node

The `SmokeParticles` node, coupled with the `SmokeGenerator` node, implements all the functionality of each smoke particle in the system. Particle splitting, as discussed in chapter 5, is implemented in this node. Furthermore, our smoke shading algorithm is also implemented in this node. Figure 6.7 is the property window of the `SmokeParticles` node. Figures in chapter 5 illustrate the visualization of smoke particles with different attribute settings of this node type. Table 6.3 describes the key attributes of the smoke particles.

Attribute	Description
mod	The period of filament generation. Reducing this number causes the generator to create filaments more frequently.
nbMotionHarmonics	The number of frequency components to use during the synthesis of each filament.
nbMotionSamples	The number of equi-distant (in the radial sense) motion samples to use for each filament. The larger this value, the more accurate the generated motion.
strength	The relative strength of the generated filament vorticities across separate current generators.
twist	The amount of twist force applied to the filaments.
thickness	The thickness of each filament. The larger this number, the greater the vorticity generated.

Table 6.1: The key attributes of a current generator.

6.3.4 `VortexRing` Node

The `VortexRing` node implements per-filament functionality and is a sub-node of the `CurrentGenerator` node type. A `CurrentGenerator` node iterates through these nodes and propagates requests, such as advection requests, control requests and display requests. The major properties of a single filament, or `VortexRing` node, are very similar to those of a `CurrentGenerator` object. An additional property, *presence*, is a toggle on whether the ring is active in the system or not. Figure 6.8 illustrates the property window of a single filament, as well as a filament rendered with (*top*) and without (*bottom*) texture sampled interpolation between motion samples (visible on the bottom right filament).

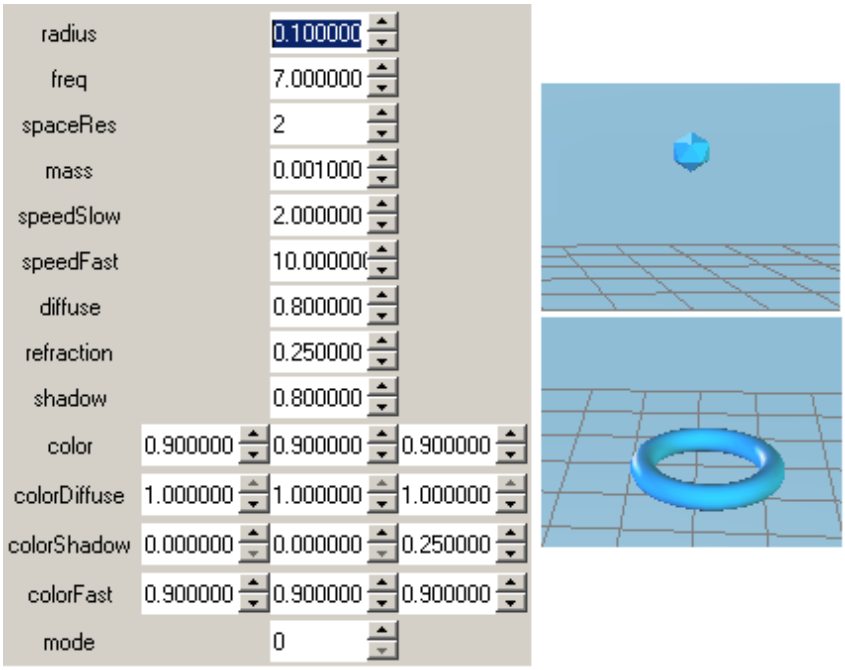


Figure 6.6: A smoke generator's property window and visualization widgets.

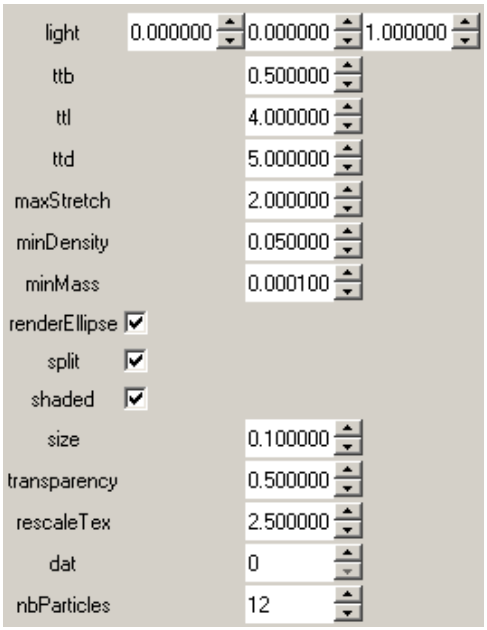


Figure 6.7: The smoke particles' property window.

Attribute	Description
radius	The size of the smoke generator. This size affects the spatial distribution of generated smoke particles. Using the gizmo to perform a uniform scale on the smoke generator widget has the same effect as modifying the radius attribute.
freq	The frequency of *particles generated per second*.
spaceRes	The sampling frequency of the distribution of particles about the cube or torus shape. For example, with the cube distribution and a spaceRes of 3, particles are spawned in a $3 \times 3 \times 3$ cube.
mass	The mass of individual particles. Particle volume is manipulated by the flow, thus adjusting the mass is an indirect method of having control over the density of smoke particles in the system.
speedSlow	Defines the *slow* speed of particles.
speedFast	Defines the *fast* speed of particles.
color	The base color of the smoke particles.
colorFast	The color of *fast* smoke particles. The final color of a smoke particle is a combination of all the color settings, including an interpolation of the base and fast colors using the speedSlow and speedFast attributes.
mode	Toggle between a cubic (mode = 0) or torus (mode = 1) distribution.

Table 6.2: The key attributes of a smoke generator.

6.3.5 `VortexSet` Node

We store various properties in an octree for acceleration purposes. This octree will be discussed in section 6.4.3. We allow the user to directly set the resolution of the octree for performance tuning. The `VortexSet` node type implements this functionality. Figure 6.9 illustrates the property window of this node type as well as a simple simulation running with octree visualization enabled. Table 6.4 summarizes the attributes of the `VortexSet`

Attribute	Description
light	The position of the point light source used during shading. As it stands, our system currently only supports one point light source.
maxStretch	The limit on the amount of stretching about the axis of maximum stretch prior to splitting a particle.
minDensity	The minimum density that must be maintained. Splitting will not occur unless the particle maintains at least this density.
renderEllipse	Toggles between rendering the smoke particle or its bounding ellipse.
split	Enables and disables particle splitting effects.
shaded	Enables and disables our self-shadowing algorithm.

Table 6.3: The key attributes of smoke particles.

node type.

Attribute	Description
renderOctree	Toggle between rendering with and without the octree grid.
octreeResolution	This value determines the degree of octree sampling and interpolation performed between leaf node samples. Varying this value allows an artist to trade performance for accuracy during design.
g	This is the definition of the global gravity field.

Table 6.4: The key attributes of the vorticity octree cache.

6.3.6 Control Node Types

We will overview the attributes of some the nodes that implement the control features of our system.

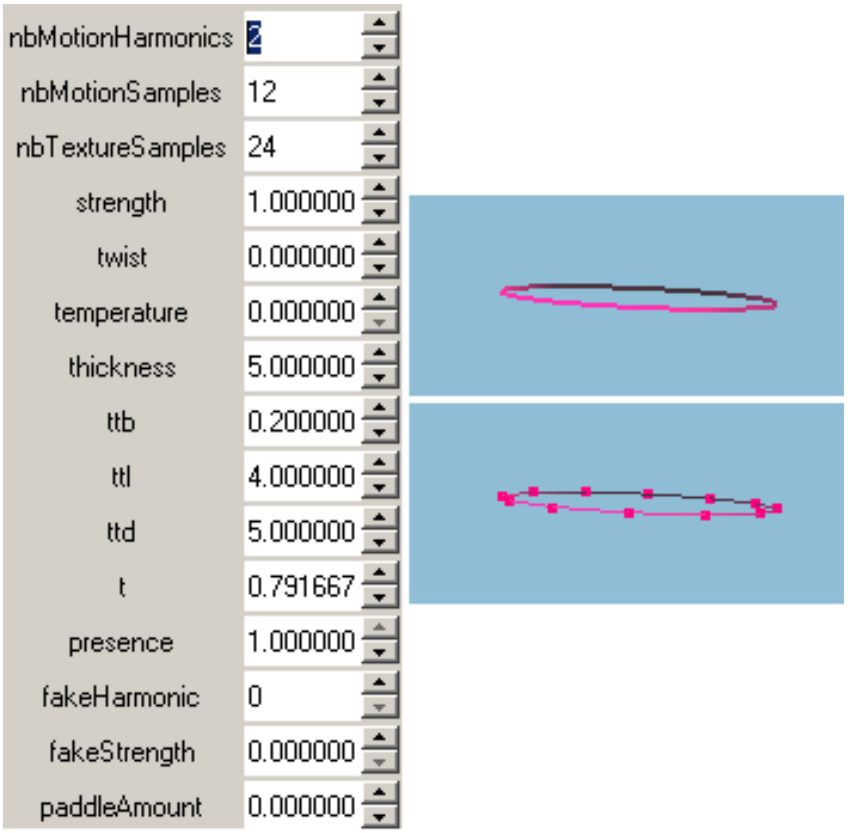


Figure 6.8: The filament's property window and visualization widget, with and without sample interpolation.

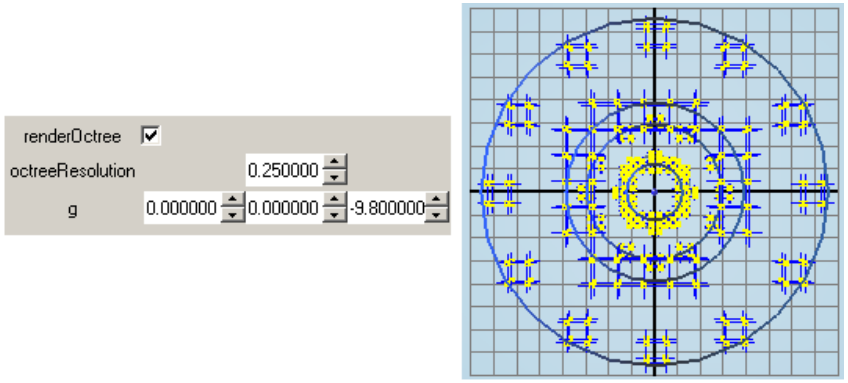


Figure 6.9: The vortex set's property window and visualization of the velocity/twist storing octree.

`CurrentAttractor` Node

The `CurrentAttractor` node type implements the functionality of the current attractor control tool. If a vortex ring is within the appropriate region of influence of the attractor, its direction and orientation are manipulated by the attractor. Figure 6.10 illustrates the property window and visualization widget of this node. Table 6.5 summarizes the key node attributes.

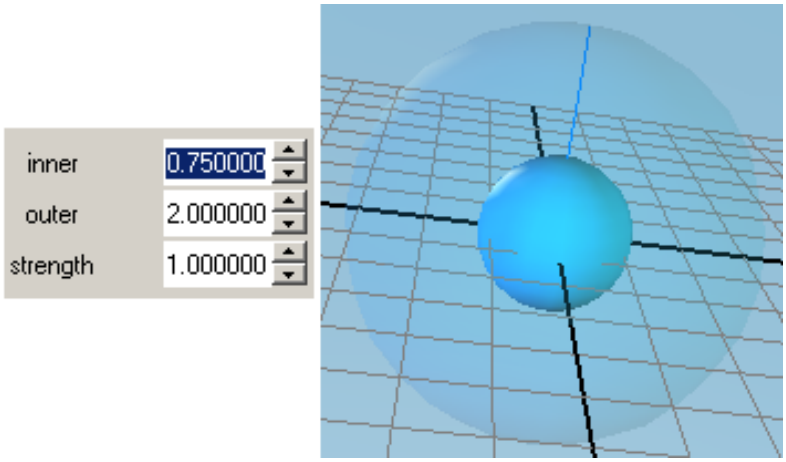


Figure 6.10: The current attractor's property window and visualization widget.

Attribute	Description
inner	The radius of the inner region of the attractor.
outer	The radius of the outer region of the attractor.
strength	This attribute affects the rate at which filaments are diverted.

Table 6.5: The key attributes of the current attractor control tool.

`CurrentTurbulence` Node

The `CurrentTurbulence` node implements the fine-level detail adding noise feature described in chapter 4. The user can specify the scale of the noise as well as the amount

of turbulence added to the system. Figure 6.11 illustrates the property window and visualization widget of the current turbulence detail tool. Table 6.6 provides a description of each of the node's key user-definable attributes.

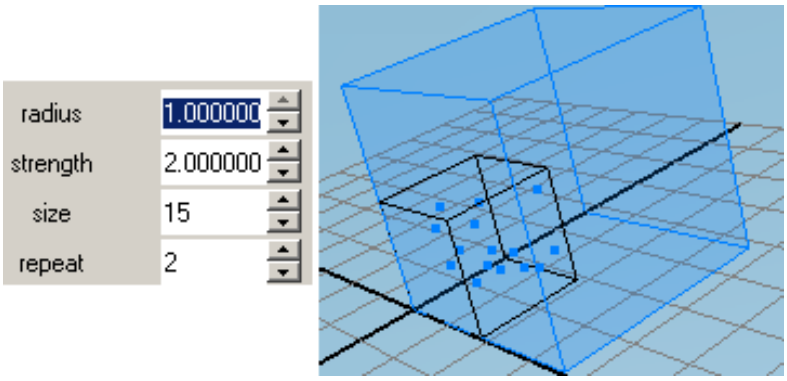


Figure 6.11: The current turbulence's property window and visualization widget.

Attribute	**Description**
size	The number of turbulent particles.
repeat	The amount of replication of the noise pattern. This value, x, subdivides the noise simulation region of the blue volume by a factor of x^2 and tiles this sub-region x^2 times.

Table 6.6: The key attributes of the current turbulence tool.

Acceleration techniques used in our implementation will be discussed in the following section.

6.4 Acceleration Techniques

One of the goals of our smoke animation system is to minimize the delay between design and visualization of results. We wish to empower our users with the ability to quickly realize whether their artistic goals are being properly reflected by the designs they implement with our system. Our smoke animation allows an artist to interact with the

smoke interactively. This is facilitated in part due to the compact representation of flow using our frequency-space analysis. The control tools operate on this basis while incurring negligible performance costs. Moreover, these tools manipulate the flow without abstracting control operations. Unlike standard fluid control mechanisms coupled with Eulerian grid techniques, the workflow of our system is suitable for rapid prototyping as well as production-quality design.

We supplement the performance advantage of our vorticity formulation with other acceleration techniques. Firstly, the GBB's node and plug architecture was designed with a data-exchange system that minimizes overhead between inter-node communication. Secondly, we accelerate the calculation of the physical properties of our system with an octree data structure. Lastly, a GPU accelerated advection engine was integrated into the GBB. We will detail these three acceleration features below.

6.4.1 Dirty Bit Updating

Recalling earlier information on GBB's plugs and nodes, node attributes are wrapped by plug objects used to communicate between attributes within a node and between separate nodes. Whenever an attribute's value is overwritten with a new value, instead of propagating the new value to all the other attributes connected to the modified attribute, and thus require them to all be updated too, the plug architecture simply propagates a *dirty bit* down the directed acyclic graph (DAG) of connected plugs. Since some attributes, such as velocities and positions, may require substantial computation for updates due to modifications of connected attributes, this dirty bit propagation mechanism can significantly reduce the performance load of our system. For example, in scenes with many interdependent nodes and constraints, such as key frames, this framework manages data transfer and reduces unnecessary computation. However, in scenes where almost all entities require updating every frame, the overhead of managing dirty bit propagation can be larger than the performance gains of avoiding the few computations that are not required.

Whenever an attribute is *read*, if its dirty bit is set due to an update of a connected attribute on a higher level in the DAG, then the attribute's value is updated and the potentially costly update routine is called only when necessary. Architecturally, this is a sound design feature of the GBB but there are some features of the GBB that may actually incur heavier performance costs than the potential benefits. For example, currently in the GBB the most common method of retrieving or modifying an attribute's value is to index the attribute by a string identifier. This operation, on a lower-level, becomes quite expensive since string comparisons must be conducted every time attribute data is accessed in the system. Our benchmarking shows that these string comparisons can combine to form the computational bottleneck of our system. Another drawback of using the GBB will be discussed shortly.

6.4.2 Hardware Acceleration

Shader Model 3.0 introduces features useful for implementing complicated shaders on graphics hardware. Of special use for physical simulations, increased shader instruction counts as well as introducing dynamic branching capabilities and native 32-bit floating point support are some key additions.

Recent advancements in the field of general programming on graphics processing units (GPUs) has motivated a hardware accelerated implementation of our advection procedure. [33] and [44] demonstrate the feasibility of accelerating physical simulations with GPUs. We will discuss the implementation of our hardware accelerated advection unit although all performance results presented in chapter 7 use an unaccelerated software implementation. For reasons that will be explained shortly, our hardware acceleration implementation is not well-suited for integration with the GBB architecture, and thus, the effect on performance due to the hardware acceleration is not fully representative of the potential this new technology can present to our specific application. However, we feel that the merits of our approach as well as our implementation can pave the way for an

implementation of our system that can maximize the benefits of a GPU implementation. We list such an implementation as one of our areas of future work in chapter 8.

Although the latest generation of GPUs are capable of achieving remarkable computational throughput, memory bus bandwidth limitations introduce performance bottlenecks.

Our hardware accelerated advection takes advantage of two fundamental benefits of our method: the compact representation of the twist transform and the straightforward vector operations required for the exponentiation of the twist. Recall from chapter 3 that our higher-order advection procedure's primary mathematical operation is the exponentiation of the twist transform. The aforementioned properties facilitate transferring the twist data over texture maps to the GPU's fragment processing unit. We choose to perform our hardware acceleration in the fragment processor since there are typically at least twice as many of these units on a standard GPU as there are vertex processing units. A pixel shader reads in the twist information and performs the advection of smoke particle and filament motion samples by taking the exponentiation of the twist transform over the desired time step.

Once the operations are complete, the updated positions are returned to the CPU for use in the next frame. Notice here that each *parallel batch* of position updates require a set of CPU-GPU data exchanges. From our experience, these data transfers are actually more costly (on a strict timing basis) than the actual operations performed on the GPU. We have investigated many methods for optimizing the data uploads and downloads between the CPU and GPU during accelerated advection. Using the OpenGL 2.0 pixel buffer object and floating point render target extensions yielded the optimal data transfer rates and minimal per-frame setup overhead. Although we have spent the majority of this chapter illustrating the benefits of the GBB architecture, it is evident that the GBB was not designed with general programming on GPU functionality in mind. Since our smoke particle advection and filament advection are performed in separate nodes intertwined

with a larger array of support nodes as well as the control nodes, we were constrained in our GPU acceleration implementation. Specifically, although the hardware accelerated advection procedure is largely identical for both smoke particles and filament motion samples, we require that two separate hardware acceleration procedures be implemented to handle the advection of these two sets of samples. In a re-engineered solution, it is plausible that a single hardware advection unit could simultaneously advect both the smoke particles and filaments in parallel. This would not only eliminate the duplicate code execution on the GPU, but would also allow us to reduce the number of CPU-GPU data exchanges by bundling the relevant twist data for both the smoke particles and filament motion samples into a single texture map. This would increase the ratio of data size to number of data transfers between the CPU and GPU, which has been shown to significantly increase the performance of a GPU accelerated procedure.

Our accelerated technique was benchmarked using an nVidia GeForce 6800 graphics card with 256 megabytes of video memory. With a minimal ratio of smoke particles and filament motion samples to CPU-GPU data transfers, the accelerated advection still performed no worse than the software simulation. As a simulation progressed and the number of particles increased, performance gains of the accelerated implementation became more apparent, however we have only noticed a maximum performance gain of approximately 8% , although this value depends heavily on the scene being simulated [5].

6.4.3 Octree Caching

One computational issue we must consider when evaluating the velocity at a point given the entire vorticity field using our Biot-Savart formulation is that for N filaments (or furthermore, $N \times M$ samples, where M is the average number of motion samples across all filaments), a naïve implementation yields an $\mathbf{O}(N^2)$ complexity for calculating all the necessary velocities of a single time step. Although this complexity is much better than the $\mathbf{O}(N^3)$ performance of naïve Eulerian uniform grid based simulators, it is still

unacceptable if we are aiming for real-time or interactive simulation performance.

We accelerate this calculation by storing the velocities (or twist transforms, for higher-order advection) in a dynamically updated octree whose grids are concentrated only around the filament motion samples and smoke particles. Velocities (or twists) are interpolated in between the octree leaf nodes. This spatial data structure reduces our time complexity to $\mathbf{O}(N)$, on average, at the cost of added memory requirements and the cost of maintaining the octree. As the number of filaments increases in a system, the benefits of this acceleration technique become more prevalent.

Furthermore, this octree mechanism provides a user with a way of fine-tuning the performance versus accuracy settings of our simulator. Recalling the attributes of the `VortexSet` node type, the resolution of the octree can be varied to increase or decrease the sampling of the data structure in exchange for decreasing or increasing the amount of (physically incorrect) interpolation of the velocities or twists used to propagate the filament motion samples and smoke particles. The performance results presented in chapter 7 illustrate the performance of our system under two separate set of system settings: one tailored for low-accuracy and fast-feedback and the other tailored at high-accuracy and slow-simulation. One of the direct contributors of this trade-off is the octree resolution used in each of these two system settings.

6.5 Summary

We introduced the open-source GBB software framework, developed by Alexis Angelidis, that we used as the foundation for our smoke animation system. Research software has a bad reputation of providing lack-luster user experiences and solely revolving around the needs of the researcher. While there are occasions when the fast-prototyping brand of software engineering is valid, since our contribution revolves around the user's experience the GBB framework was well-suited to our needs.

The powerful yet easy-to-use interaction components of the GBB were overviewed, its underlying data transfer architecture as well as its large support of common computer graphics and computer animation functional units were also summarized.

In a sense, despite the professional and robust architecture of the GBB and its strength through generality, it is this general computer graphics development platform design that also somewhat limits the full performance potential of our smoke simulation and animation system. With careful planning, a more suitable and specially-tailored implementation of our system will allow the full benefits of GPU acceleration to be attained. A more optimized data transfer mechanism, as discussed earlier in this chapter, will also likely yield further performance gains. The UI themes and features of the GBB will most likely remain, in one form or another, given time to re-engineer a new system.

Our smoke animation system was implemented as an extension to the GBB comprised of multiple additional components. The key simulation and control components were overviewed as well as their visualization entities in the GBB and the main attributes of these components exposed to the user. We ended with a summary of the performance-tailored features of the GBB and our implementation.

Chapter 7

Results

We will present results, feedback and performance measurements of our smoke animation system in this chapter. Firstly, final rendering frames from animation sequences we have generated with the system will be presented. Since the effect of varying smoke particle settings on final rendered images was presented in chapter 5, we will focus on effects based solely on the control of filaments in our system.

7.1 Simulation Renderings

The final renderings are presented in this section to illustrate some of the different effects, both artistic and functional, an artist can incorporate into an animation sequence using our method.

7.1.1 Control Curve Effects

A control curve is used in the scene below to prevent the smoke from obscuring the character's face and to bring the smoke into view after the view orientation and position have changed.

Figure 7.1: Using a current control curve to move the smoke into an artistically required location [5].

Moreover, the following two animation clips use current control curves to smother a fly with a toxic gas.

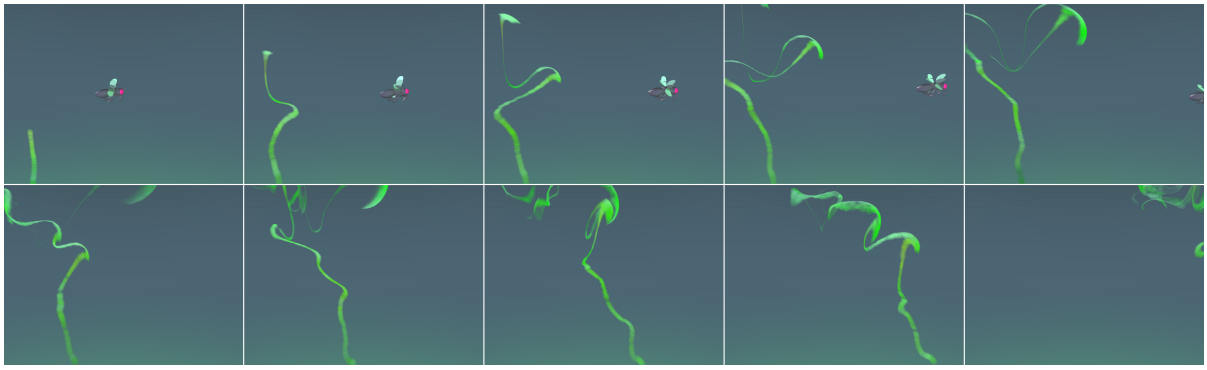


Figure 7.2: Toxic smoke chasing the fly [5].

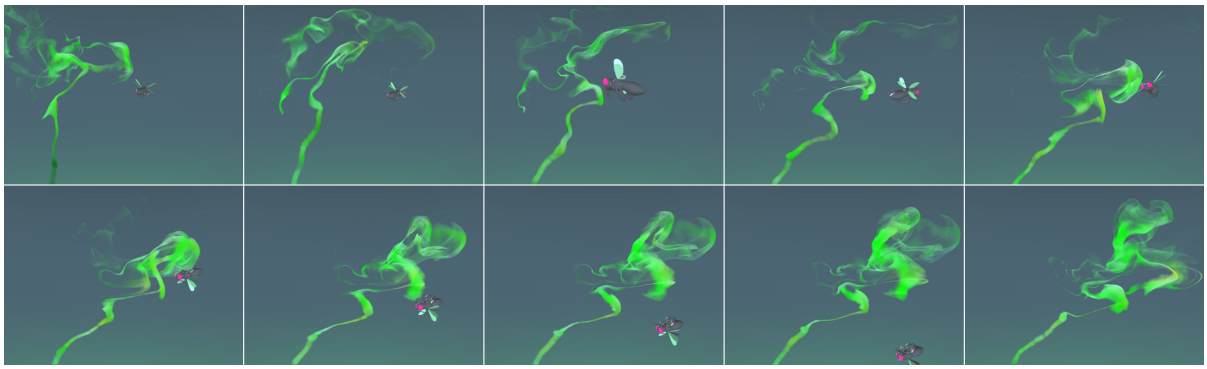


Figure 7.3: Toxic smoke catching the fly [5].

The control curve is illustrated and particle splitting is turned off to exaggerate how a simple scene setup can yield convincing results.

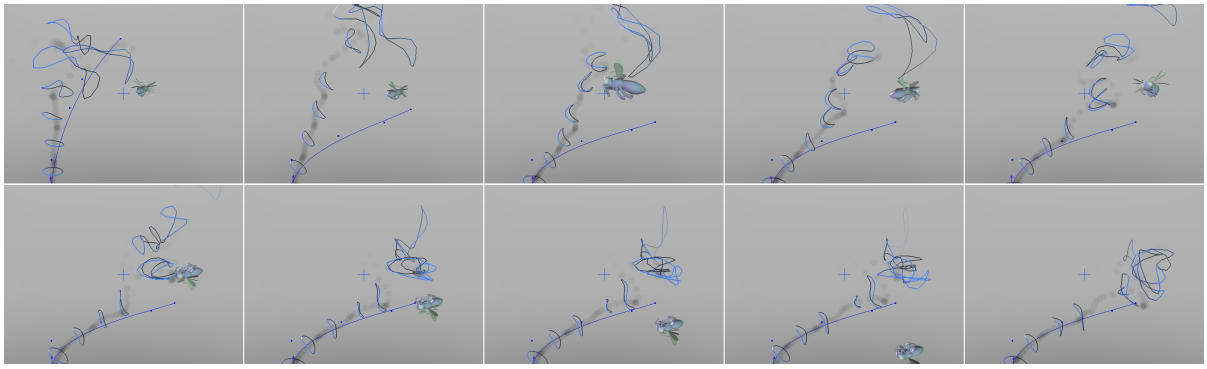


Figure 7.4: The design view of the fly scene [5].

7.1.2 Current Attractor Effects

Current attractors are used in the following animation to ensure that a layer of smoke always remains under the genie.

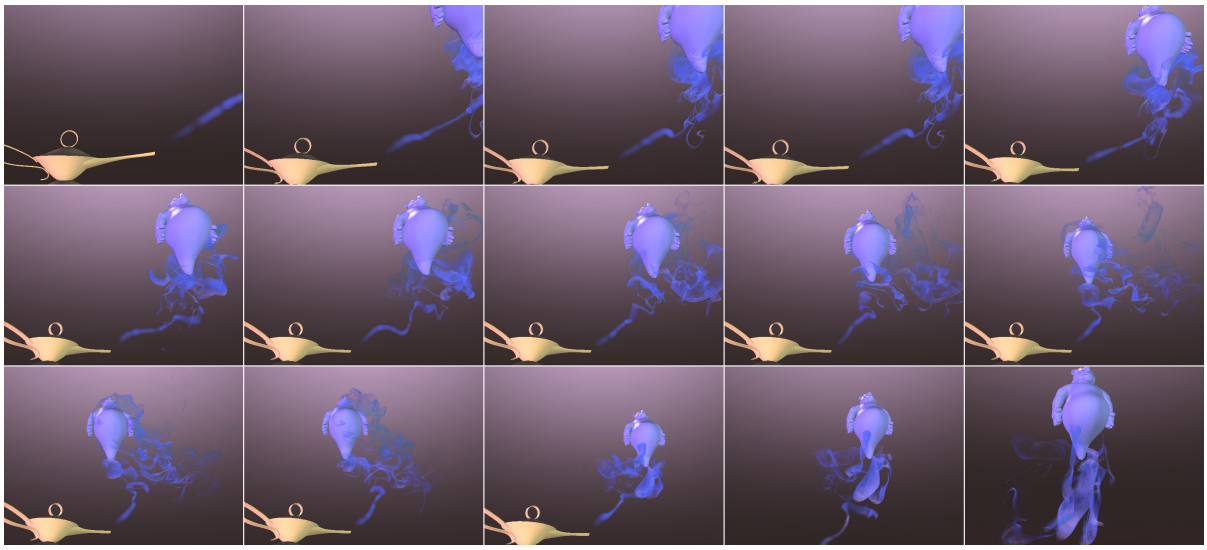


Figure 7.5: A genie's lamp smoke follows him throughout his flight [5].

7.1.3 Tornado Effects

The twist feature can create some very interesting effects at low computational cost thanks in part to our compact filament basis. The following animation sequence illustrates the twisting effect on a swirl of three different smoke strands.

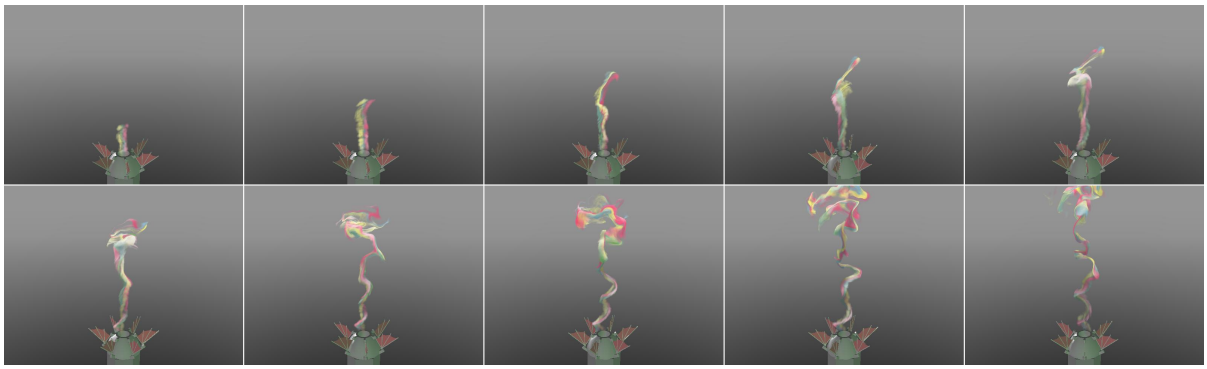


Figure 7.6: Smoke swirling about an axis along its path [5].

7.1.4 Fine-level Detail Effects

The following sequence illustrates the effects of adding noise vortices into the flow. The top row is the sequence *without* the added noise vortices and the bottom row is the sequence *with* the noise added.

Figure 7.7: The effect of noise vortices for adding fine-level detail [5].

7.2 User Experience Feedback

We have distributed our animation system to a set of computer graphics aware users with novice and intermediate animation expertise. With little to no instruction, the users were able to generate simple animation sequences in a short period of time. Adding more complicated effects to the animation sequences was found to be straightforward, from a user perspective, once certain advanced usage techniques were explicitly demonstrated.

Users found our system's interface intuitive despite its simplicity. Including more common functionality as buttons or widgets outside of the menuing layout was a secondary request. Scene entity layout, key framing and interaction were easily grasped by the users with prior animation experience. Users were particularly impressed with the responsiveness of our system and the quality of their final rendered animation sequences.

Apart from this informal user study, we have analysed the key reviewer comments from

anonymous academic experts in the fields of computer graphics, computer animation and applications of computational fluid dynamics in computer graphics. Comments spanned the key areas of our contribution: our filament basis, our control tools and our rendering algorithm.

The comments on our compact filament basis from the reviewers were generally positive. For the most part, reviewers understood the benefits of our representation and those who made the connection to the relevance of this representation for filament control were especially impressed. Although our frequency space analysis does contribute toward the speed improvement of our system over previous systems as well as providing improved sampling, the main benefit of our representation is its connection to the control tools.

The quality of the screenshots of our final animation sequences drew the attention of most of our audience. Considering the fact that our rendering algorithm is computational efficient and not specially designed for feature-film quality productions, the feedback we received was pleasant.

Furthermore, a handful of experts were very impressed by our system and the results we presented and suggested that our technique be incorporated into current production pipelines as well as educational programs for computer graphics and fluid dynamics. Moreover, some reviewers could not distinguish whether our renderings were performed in real-time or using a ray-tracing method.

7.3 Runtime Performance

The performance of our method is measured in tables 7.1 and 7.2 using sample animation sequences generated with our animation system. We measure the performance of our system using two sets of system settings. *Modeling* settings are a combination of system setting values set with the purpose of reducing computational cost. These values are set while trying to balance a level of accuracy close to the more physically-accurate results of

our higher-quality simulations and renderings. The *high-quality* settings include effects such as noise, adaptive particle splitting and a higher number of filament basis component frequencies used during simulation. Furthermore, the number of filament motion samples and the number of smoke particles are increased to properly sample the visual space. Lastly, the octree resolution is reduced in the modeling settings in order to increase the responsiveness of our system. Modeling settings are meant to represent the settings an artist would use during the animation design phase. High-quality settings would then be used during the refinement and final rendering phases of the production.

The complexity of a scene is measured with the average number of filament particles per frame, the average number of smoke particles per frame and whether the scene takes advantage of particle splitting and/or noise effects. The performance is measured as the *average* frame rate over a 200 frame long animation.

Scene	Modeling Settings		
	Avg. # of Motion Samples	Avg. # of Particles	Avg. # of FPS
Happy Boy	113	72	21.71
Simple Benchmark	78	96	18.43
Toxic Fumes vs. the Fly	45	112	18.32
Noise Benchmark	91	112	12.63
Genie & Lamp	67	2493	12.435
Forest Fire	120	4875	5.16
Walkthrough Smoke	395	1280	4.744

Table 7.1: Benchmarks with modeling settings settings.

Scene	High-Quality Settings				
	Particle Splitting	Noise	Avg. # of Motion Samples	Avg. # of Particles	Avg. # of FPS
Happy Boy	✓		450	1197	3.61
Simple Benchmark			90	168	7.63
Toxic Fumes vs. the Fly	✓	✓	182	3787	0.37
Noise Benchmark	✓	✓	364	3038	2.516
Genie & Lamp	✓	✓	450	73357	0.24
Forest Fire		✓	336	20250	0.54
Walkthrough Smoke	✓	✓	750	7040	1.03

Table 7.2: Benchmarks with high-quality settings.

The results in Tables 7.1 and 7.2 illustrate the linear scalability of our method in the number of particles and filaments. Interactive framerates are achieved in all modeling settings simulations, and some less demanding high-quality settings simulations.

7.4 Summary

We have presented screenshots of some animation sequences modeled with our system, demonstrating examples of different effects that can be designed using our system's control tools. Every animation sequence was modeled in under two hours, some in under one hour, by an artist with intermediate to advanced level knowledge of our system. Furthermore, the simulation and rendering time of the final animation sequences for all the scenes combined took less than eight hours, demonstrating a significant reduction in the time required to design and yield final animation results when compared to standard

Eulerian grid-based techniques.

The general comments from a pool of student volunteers as well as the technical comments from anonymous academic experts were presented in this chapter, followed by performance data captured on a set of animation sequences we designed using our system. The performance measurements illustrated the scalability of our system as well as the ability of providing an artist with a multi-level editing framework.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

We have presented an animation system that allows an artist to manipulate both thin and thick smoke with tools that are familiar to computer graphics content designers. We use a vortex method simulation engine to represent the flow field used to advect visualization particles. Vorticity filaments are the main flow inducing entities of our physical simulation and we perform a frequency-component harmonic decomposition on the filaments. This novel decomposition allows an artist to animate smoke with a level-of-detail representation previously unavailable in a single pass of a simulation. Our decomposition is also used to provide a basis for controlling the filaments. The path of the filaments can be constrained without jeopardizing the physical integrity of the system in a manner that has not previously been investigated for fluid control.

The main advantage of our system over previous high-quality fluid animation packages is that we enable an interactive manipulation of smoke parameters during simulation as well as real-time visualization of the phenomenon, eliminating the typical delays in previous design pipelines. Level-of-detail filament representations allow an artist to design an animation with a simulation engine that responds in real-time. Intuitive control

mechanisms, such as control curves, allow for powerful control over the flow without over-complicated and restrictive interfaces.

Our system is built on top of a robust, open-source computer animation software architecture with a straightforward application programming interface (API) that easily allows additional functionality to be added. A testament to the usability of our software is that a novice user can create a relatively complicated animation sequence with little or no training in under an hour. All of the animation sequences created for this thesis and for [5] were created in under two hours.

8.2 Future Work

We foresee potential areas of future work and research focusing on the physical simulation component of our system, different rendering models for our system and further acceleration of our system.

8.2.1 Physics Future Work

We can extend the capabilities of our physical simulator by adding some of the terms we omitted in the vorticity transport equation. Recently, computer graphics researchers have extended standard grid-based simulators with support for viscous fluids [30] or have devised ad-hoc simulations of viscous materials [15]. If we reformulate the vorticity transport equation to include the viscous diffusion term $\nu\nabla^2\vec{w}$ and devise an efficient method of incorporating these viscosity effects into our current framework, we may be able to create interesting effects involving smoke and the effects of viscosity.

Viscosity is a more compelling effect when applied to liquids. Thus it follows that another significant area of future research would be the simulation of liquids with vortex methods. Since the usefulness of filament primitives as flow carrying elements only pertains to the simulation of smoke, an alternative vortex representation would be necessary

for the simulation of liquids. Applying a similar methodology for the simulation of liquids as we did for our simulation of smoke may lead to an efficient and intuitive artistic workflow for the design of liquid animations. For example, the direct manipulation of the liquid-air interface or indirect flow generation with control tools behaving as physical objects in the system are interesting ideas that we will investigate. Overlapped vortex sheets or vortex particles are a few flow primitives we will consider using.

Stepping back into the realm of smoke simulation, since our physical simulation only takes the effects of advection into account, it would be interesting to introduce the fine level effects of diffusion into our simulation. Perhaps a sub-simulation at the per-particle level could introduce a finer scale detail. Our noise features are a compromise between very-fine level detail control and absolutely no fine level detail control. Having a separate smoke particle sub-simulation to control the fine-level details could add even more detailed realism to our smoke simulations.

The area of object boundary handling with vortex methods still could use improvement and if an efficient technique for handling collisions of our smoke particles or current filaments with *arbitrary* objects or a subset of objects (such as spheres or convex objects) would be an interesting and potentially rewarding area of research with applications that could move into the computational fluid dynamics research community. Furthermore, once arbitrary object boundary effects are feasible without substantial performance loss, integration of a real-time controllable smoke simulator with existing real-time rigid body simulators could yield fully-coupled gas-solid interactive simulations for use in gaming and animation applications.

Lastly, in chapter 4 we introduced our novel and compact filament basis. Investigating other compact and robust methods of representing filaments may lead to further accelerations and other unforeseen benefits.

8.2.2 Rendering Future Work

Although our system currently focuses on real-time realistic rendering as its primary rendering solution, two additional areas of interest for future work are high-fidelity physically-based offline rendering algorithms and artistic rendering algorithms. Needless to say, more sophisticated and effective rendering algorithms are also still of interest for future research directions.

Recent advances in real-time rendering algorithms have focused on techniques that perform precomputations and domain transformations on the standard rendering equation in hopes of reproducing complex lighting effects, such as diffuse interreflection, in real-time. Traditionally, the computational cost of calculating these complicated global illumination effects precluded their application in real-time rendering algorithms. *Precomputed Radiance Transfer* (PRT) techniques are evolving to potentially solve this problem. Although the initial focus of PRT techniques was on relighting static scene geometry while addressing transport issues involved in compactly representing the view-dependent nature of many complicated lighting effects, the research focus has shifted slightly towards continuing to represent complicated effects *without* assuming a fixed scene geometry.

Ren *et al.* have recently described a PRT algorithm that aims to accurately capture soft shadowing effects of dynamically moving objects on themselves and their neighbouring environment [66]. As with other PRT techniques, concurrent viewing and incoming lighting direction changes can be made without affecting the performance of their system. This work is of particular interest for the purposes of smoke rendering. Ren *et al.* approximate the objects in a dynamic scene as a hierarchical combination of spheres and use this approximation, along with a novel log space visibility accumulation technique, to dynamically shadow the scene in real-time. If a similar technique can be adopted to handle attenuations amenable to a gas, such as smoke, as opposed to solid objects, then our current ellipsoidal smoke representation can be leveraged with such a technique for more realistic real-time rendering effects.

Another potential area of future work in rendering is non-photorealistic rendering algorithms for smoke. Originally, Selle *et al.* coupled a billboarding algorithm along with depth buffer differencing and standard cartoon rendering techniques to allow for cell-shaded smoke animations generated from grid-based smoke simulations [71]. Recently, McGuire and Fein followed up with a cartoon smoke renderer with billboarding and efficient silhouette outlining as well as a novel self-shadowing algorithm based loosely on real-time shadow volume methods [52]. Once again we may be able to leverage our current particle representation. Our particle representation allows us to easily extract and sort depth information (our current rendering algorithm already performs these calculations in real-time) and the *displacement gradient tensor* information may be useful when applied to artistic and stylized smoke rendering methods.

Lastly, our system can be extended with the incorporation of an offline physically-based volumetric rendering algorithm similar to previous works presented by Premoze *et al.* in the area of light transport in participating media [63]. Ideally we can leverage the smoke particles of our system to either simplify the domain of integration of the volumetric rendering equation or to accelerate the calculations.

8.2.3 Future Work in Acceleration and Software Engineering

As mentioned earlier, our system is implemented as an extension to the *Graphics Blue Box* software toolkit. While the benefits of this toolkit were outlined, it was designed with generality in mind and thus contains additional and possibly unnecessary layers of abstraction. One future area of work would be to re-write our system as a stand-alone software package, thus eliminating any excess software abstraction. Such a re-engineering would also allow us to couple all hardware acceleration units together, in hopes of improved performance, instead of having separate hardware acceleration units. This change makes sense because while the calculations for advecting filaments and particles are almost identical, in our current system, they are executed as a sequential batch of parallel

operations on the GPU, as opposed to one parallel operation. The Graphics Blue Box also identifies internal variables as string types, and thus a string comparison operation is performed for each variable query. During benchmarking, it was noted that these string comparison operations consume a significant amount of CPU processing during execution.

Recently, a sub-area of the *General Programming on Graphics Processing Units* (GPGPU) research area has focused on the implementation of data structures on GPUs. This task is especially challenging on the current generation of GPUs since random access writes to graphics memory are not supported and random access reads from graphics memory are prohibitively expensive to perform on a scale necessary for standard data structure implementations. Lefebre *et al.* implemented octree data structure support on GPUs with an application for three dimensional texturing [47]. Recently, Lefohn *et al.* [48] extended the work of Kniss *et al.* [43] with a robust and comprehensive generic data structure library for GPUs. Extending the hardware acceleration of our system, namely our octree acceleration, with one of the existing packages or a tailor made data structure on GPU software package is another area of potential future work.

8.3 Final Thoughts

Our system can create cinematic quality animations of wispy smoke in a fraction of the time current Eulerian simulators require while providing the artist with design tools and response rates that further accelerate the complete design cycle of an animation sequence. Test users and academic experts were impressed with the results of our system and the simulation performance compared to standard Eulerian techniques. Hopefully with time and exposure, our method will contribute to future techniques that balance high-quality results with fine-tuned workflow improvements.

Other Lagrangian schemes for representing gaseous flow should be validated against

our approach. The use of vortex methods that use different discretization primitives, such as vortex sheets or vortex particles, provide good numerical and visual bases for comparison with our technique. Implementing a vortex sheet simulator based on our system is a natural extension that can be used during this comparison. The work of Park and Kim [58] is also a good candidate for comparison since they also consider the same speed and accuracy trade-offs as our system.

A book is never finished, it is only published. – Derick Wood

Bibliography

[1] Alexis Angelidis. *Shape Modeling by Swept Space Deformation*. PhD thesis, University of Otago, Dunedin, New Zealand, 2005.

[2] Alexis Angelidis. Personal communication, 2005–2006.

[3] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: Constant volume modeling. In *Pacific Graphics*, Korea, October 2004. Best paper award.

[4] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*. ACM Press / ACM SIGGRAPH / EuroGraphics, 2005.

[5] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *ACM-SIGGRAPH/EuroGraphics Symposium on Computer Animation (SCA)*. ACM Press / ACM SIGGRAPH / EuroGraphics, September 2006.

[6] Alexis Angelidis, Derek Nowrouzezahrai, and Patricio Simari. Interactive modeling and rendering of second order adaptive patch surfaces. In *In Progress*, pages 1–8, 2006.

[7] H. Anton. *Calculus: A New Horizon*. John Wiley and Sons, 1998.

[8] J. E. Barnes and P. Hut. A hierarchical o (n log n) force calculation algorithm. *Nature*, 324:446, 1986.

[9] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[10] William Brennan. Smoke abstractions photo gallery, 2005. http://www.pbase.com/billyb2/smoke.

[11] N. Chiba, K. Muraoka, H. Takahashi, and M. Miura. Two-dimensional visual simulation of flames, smoke and the spread of fire. *Journal of Visualization and Computer Animation*, 5(1):37–54, January "–" March 1994.

[12] A. J. Chorin. Vortex models and boundary layer instability. *SIAM J. Sci. Stat. Comput. 1*, pages 1–21, 1980.

[13] Derek Nowrouzezahrai Christian Lessig and Karan Singh. Gpu-accelerated ray casting of node-based implicits. *ACM SIGGRAPH 2006: Poster session of the 33rd annual conference on Computer graphics and interactive techniques*, 2006.

[14] J. P. Christiansen. Numerical simulation of hydrodynamics by the method of point vortices. *Journal of Computational Physics*, 13:363–379, 1973.

[15] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Symposium on Computer Animation 2005*, pages 219–228, July 2005.

[16] G-H Cottet and P D Koumoutsakos. Vortex methods: Theory and practice. *Measurement Science and Technology*, 12(3):354, 2001.

[17] D. S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline a-buffer techniques. In *SIGGRAPH '90:*

Proceedings of the 17th annual conference on Computer graphics and interactive techniques, pages 357–366, New York, NY, USA, 1990. ACM Press.

[18] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Transactions on Graphics*, 23(3):441–448, 2004.

[19] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In Eugene Fiume, editor, *ACM SIGGRAPH 2001: Proceedings of the 28th annual Conference on Computer Graphics Proceedings and Interactive Techniques*, pages 15–22. ACM Press / ACM SIGGRAPH, 2001.

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

[21] Bryan E. Feldman, James F. O'Brien, Bryan M. Klingner, and Tolga G. Goktekin. Fluids in deforming meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005*, July 2005.

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

[23] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

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

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

[26] J. G. Russo and A. Strain. Fast triangulated vortex methods for the 2d euler equations. *Journal of Computational Physics*, 111:291–323, 1994.

[27] J. Gallier. *Geometric Methods and Applications: For Computer Science and Engineering*. Springer, 2000.

[28] F.J. Gerstner. Theorie der wellen. *Ann. der Physik*, 32:412–440, 1809.

[29] A. Gharakhani and A. F. Ghoniem. Three-dimensional vortex simulation of time dependent incompressible internal viscous flows. *Journal of Computational Physics*, 134:75–95, 1997.

[30] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. A method for animating viscoelastic fluids. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2004 Conference on Computer Graphics and Interactive Techniques)*, 23(3):463–468, 2004.

[31] J. R. Grant and J. S. Marshall. Inviscid interaction of vortex rings: approach to singularity? *ESAIM Proceedings*, 7:183–194, 1999.

[32] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 1987:325–348, 1973.

[33] Eitan Grinspun, Jeff Bolz, Ian Farmer, and Peter Schrder. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. *SIGGRAPH (ACM Transactions on Graphics)*, 22(3), 2003.

[34] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *ACM SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1993. ACM Press.

[35] F. Harlow, J. Shannon, and J. Welch. The mac method: A computing technique for solving viscous, incompressible, transient fluid-flow problems involving free surfaces. Los Alamos, NM, USA, 1965. Los Alamos Scientific Laboratory.

[36] Clay Mathematics Institute. Millennium prize problems, 2006. http://www.claymath.org/millennium/.

[37] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *ACM SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320, New York, NY, USA, 1998. ACM Press.

[38] I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.

[39] James T. Kajiya. The rendering equation. In *ACM SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM Press.

[40] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *ACM SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1984. ACM Press.

[41] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM Press.

[42] Arie Kaufman and Eyal Shimony. 3d scan-conversion algorithms for voxel-based graphics. In *SI3D '86: Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 45–75, New York, NY, USA, 1987. ACM Press.

[43] Joe M. Kniss, Aaron Lefohn, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Octree textures on graphics hardware. In *ACM SIGGRAPH 2005 Conference Abstracts and Applications*, August 2005.

[44] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131, New York, NY, USA, 2004. ACM Press.

[45] Hess J. L. and Smith A. M. Calculation of non-lifting potential flow about arbitrary three-dimensional bodies. *Journal of Ship Research*, 8:22–44, 1964.

[46] Arnauld Lamorlette and Nick Foster. Structural modeling of flames for a production environment. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 729–735, New York, NY, USA, 2002. ACM Press.

[47] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter Octree Textures on the GPU, pages 595–613. Addison Wesley, 2005.

[48] Aaron Lefohn, Joe M. Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Glift: Generic, efficient, random-access gpu data structures. *ACM Transactions on Graphics*, 25(1):60–99, January 2006.

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

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

[51] Daniel Margerit. Mouvement et dynamique des laments et des anneaux tourbillons de faible epaisseur. *PhD thesis*, 1997.

[52] Morgan McGuire and Andi Fein. Real-time cartoon rendering of smoke and clouds. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, June 2006.

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

[54] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.

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

[56] S. Osher and R. Fedkiw. *Level set methods: an overview and some recent results.* Springer, 2000.

[57] M. L. Ould-Salihi, G.-H. Cottet, and M. El Hamraoui. Blending finite-difference and vortex methods for incompressible flow computations. *SIAM Journal of Scientific Computing*, 5:1655–1674, 2000.

[58] Sang Il Park and Myoung Jun Kim. Vortex fluid for gaseous phenomena. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 261–270, New York, NY, USA, 2005. ACM Press.

[59] Ken Perlin. An image synthesizer. In *ACM SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press.

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

[61] Xavier Pintado and Eugene Fiume. Grafields: Field-directed dynamic splines for interactive motion control. *Computers & Graphics*, 13(1):77–82, 1989.

[62] Jovan Popovic, Steven M. Seitz, Michael Erdmann, Zoran Popovic, and Andrew Witkin. Interactive manipulation of rigid body simulations. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 209–218. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[63] Simon Premoze, Michael Ashikhmin, Jerry Tessendorf, Ravi Ramamoorthi, and Shree Nayar. Practical Rendering of Multiple Scattering Effects in Participating Media. In *EuroGraphics 2004 Symposium on Rendering*, number 2004, 2004.

[64] W. J. W. Rarddne. On the exact form of waves near the surfaces of deep water. *Phil. Trans. Roy. Soc.*, A 153:127–138, 1863.

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

[66] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics*, 25(3):977–986, 2006.

[67] L. Rosenhead. The formation of vorticies from a surface of discontinuity. *Proc. Roy. Soc. London Ser.*, A 134:170–192, 1931.

[68] A. Rutherford. *Vectors, Tensors, and the Basic Equations of Fluid Mechanics*. Dover Publications, Inc., 1989.

[69] Elcott S., Tong Y., Kanso E., Schrder P., and Desbrun M. Discrete, circulation-preserving, and stable simplicial fluids. *Preprint*, pages 1–61, 2005.

[70] Georgios Sakas. Modeling and animating turbulent gaseous phenomena using spectral synthesis. *Vis. Comput.*, 9(4):200–212, 1993.

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

[72] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–236, New York, NY, USA, 2005. ACM Press.

[73] Mikio Shinya and Alain Fournier. Stochastic motion-motion under the influence of wind. In *Computer Graphics Forum, Volume 11, Issue 3*, pages 119–128, 1992.

[74] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

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

[76] J. Steinhoff and D. Underhill. Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6:2738–2744, August 1994.

[77] Mark J. Stock. Summary of vortex methods literature (a living document rife with opinion), 2005.

[78] D. M. Summers and A. J. Chorin. Hybrid vortex/magnet methods for flow over a solid boundary. *ESAIM Proceedings*, 1:65–76, 1996.

[79] W.Y. Tan. *Shallow Water Hydrodynamics*. Elsevier, 1992.

[80] Adrien Treuille, Antoine McNamara, Zoran Popovic, and Jos Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics*, 22(3):716–723, 2003.

[81] K. Voss. Discrete images, objects, and functions in zn. volume "11", Berlin, Heidelberg, New York, 1993. Springer-Verlag.

[82] Jakub Wejchert and David Haumann. Animation aerodynamics. In *ACM SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 19–22, New York, NY, USA, 1991. ACM Press.

[83] H. S. Wilf. The method of characteristics, and "problem 89" of Graham, Knuth and Patashnik. *ArXiv Mathematics e-prints*, June 2004.

[84] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *ACM SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, New York, NY, USA, 2001. ACM Press.