Efficient Algorithms for Controllable Fluid Simulations and High-Resolution Level Set Deformations

Brian Bunch Christensen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Efficient Algorithms for Controllable Fluid Simulations and High-Resolution Level Set Deformations

A Dissertation
Presented to the Faculty of Science
of Aarhus University
in Partial Fulfillment of the Requirements for the
PhD Degree

by
Brian Bunch Christensen
March, 2010

ABSTRACT

During the past decade, we have seen a rapid escalation in the amount and scope of computerized visual effects employed in feature films. This trend is in large due to the advances of computational techniques for doing physically based animation. Level sets, which are dynamic implicit surfaces developed for computational physics, are a prime example of such a technique, that has found use in computer graphics, computer vision and visual effects. Within computer graphics they have been used as the underlying surface representation in water simulations, geometric modeling, shape metamorphosis and several other applications. Among the most resource-demanding effects in today's feature films is the simulation of smoke and water. As a result, level sets and their accompanying technology are constantly being pushed to the limits. Besides being computationally expensive, the unpredictable behavior of the simulations and lack of explicit artistic control result in a workflow marred by trials and errors.

In the first segment of this dissertation, I propose an out-of-core framework for efficient streaming of level set computations. It is motivated by the fact that increasingly higher resolutions are desired in all level set application areas, and that disk space in general offers much higher storage capacity and is two to three orders of magnitude cheaper than internal memory. The framework is built on top of an existing out-of-core framework and the new contributions include code transformations which decrease the number of times data must be streamed from disk during simulation. As a result, the framework is now CPU limited and can sustain a throughput of up to 92% of in-core simulations on desktop computers with limited memory resources even for simulations requiring several gigabytes of storage. Furthermore, the code transformations allow for parallel processing of individual tiles of the level set which also reduces the computation time required.

The second segment focuses on efficient algorithms for controlling the motion of gaseous fluid phenomena. Normally a simulation is very sensitive to the resolution of the simulation domain, and often the final behavior is changed completely when the resolution is modified. I propose a control framework for smoke animations which couples simulations of different resolution together in a way such that their bulk motions are similar. The control framework ideally allows a visual effects artist to tune simulation parameters using a coarse, faster prototype simulation and then use that simulation to *guide* a higher resolution simulation which has the desired amount of details, but the same overall behavior.

ACKNOWLEDGEMENTS

First of all I want to thank my advisor Ken Museth for his guidance and support. He has opened my eyes to the exciting fields of level set methods and fluids and has been a source of inspiration for me. Because of the geographical distance and time difference, he has devoted many off-hours to answering my questions and providing expert advice on how to proceed. Furthermore, I thank him for initiating and handling the contact to Digital Domain, where I spent three exciting and enlightening months. Secondly, I acknowledge my advisor Susanne Bødker at Aarhus University. I am grateful to her for providing advice on doing research, and for encouraging me to pursue a PhD degree. Thirdly, I am indebted to Michael Bang Nielsen for providing ideas and inspiration, and for answering my many questions on a daily basis. He has truly been my third, unofficial "advisor", and I would like to emphasize that without his and Ken's guidance the work presented in this dissertation would not have been possible.

Additionally, I thank the people of Digital Domain for giving me an opportunity to experience the exciting world of visual effects and to see how computer graphics can be applied in practice. It was a great professional and personal experience. In particular, I wish to thank Doug Roble and Nafees Bin Zafar for collaborating with me on the fluid control framework, and Michael Clive and Ryo Sakaguchi for helping define the control problem. I generally thank all of my collaborators for many inspiring discussions.

I would like to credit Ken Museth, Michael Bang Nielsen, Susanne Bødker, Clemens Nylandsted Klokmose and Anja Hovgaard for commenting on drafts of this dissertation. I furthermore thank Clemens Nylandsted Klokmose for providing an excellent LaTeX template.

Finally, I thank Anja for enduring the writing process with me and encouraging me to hang in there. I am grateful to her and my parents Connie and Hans for their love and support.

The research presented in this dissertation was funded by the Faculty of Science at Aarhus University.

Brian Bunch Christensen,
Aarhus, March, 2010.

PREFACE

This dissertation is divided into two parts: The first part provides a coherent overview of the work of this PhD project as well as a more in-depth discussion of the background material and related work. The second part is a collection of two published conference papers and one submitted journal paper.

The overview is divided into eight chapters. Chapter 1 is an introductory chapter that motivates the research, briefly introduces the visual effects pipelines, provides an overview of the central contributions, and outlines the rest of the dissertation. Chapters 2 and 3 serve as an introduction to level sets and the level set method that form the basis upon which the submitted journal paper is built. In particular, chapter 2 presents the original level set method whereas chapter 3 looks at relevant extensions and improvements. Chapter 4 supplements paper I by more thoroughly explaining some of the concepts and background. Next, chapters 5 and 6 change focus to fluid simulations in the context of computer graphics. This acts as a prelude to the work presented in the two published papers. Chapter 5 serves as an introduction to fluid simulations and explains a basic method for simulating fluids in visual effects, while chapter 6 presents several improvements. Chapter 7 motivates the need for a method for controlling fluid simulations, such as the control framework of papers II and III, and provides a discussion of a number of other control methodologies. Finally, chapter 8 concludes the overview and discusses future work.

The recommended guideline for reading this dissertation is as following: Start with chapters 1 – 4 for an introduction to the entire dissertation in general and the work on high-resolution level sets in particular. Then read paper I. Continue with chapters 5 – 7 for an overview of the work on controlling fluid simulations. Lastly, read papers II and III and conclude with chapter 8.

Video material from the papers mentioned above is available on the CD-ROM accompanying this dissertation.

INCLUDED PAPERS

[1] Brian B. Christensen, Michael B. Nielsen, and Ken Museth. Out-of-core computations of high-resolution level sets by means of code transformation. *Journal of Scientific Computing*, 2010. Submitted.

[2] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolution. In *ACM SIGGRAPH Symposium on Computer Animation 2009*, pages 206–215, August 2009.

[3] Michael B. Nielsen and Brian B. Christensen. Improved variational guiding of smoke animations. In *Computer Graphics Forum / Proceedings of the Eurographics Conference 2010*, pages x–$(x+7)$, May 2010. To appear.

[4] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. A variational framework for guiding of smoke animations. 2010. In preparation. Segments included in chapter 7.

CONTENTS

CHAPTER 1

INTRODUCTION

The main topic of this dissertation concerns level set and fluid simulations for computer graphics. A method is proposed for efficiently performing computations on high-resolution level sets that do not fit in main memory. In brief, the research presented transforms the level set algorithms to require only a single pass over the data. Also, a novel approach to controlling high-resolution fluid simulations based on coarse prototypes is proposed and explored.

In recent years, we have seen an enormous escalation in the amount and sheer scope of computerized special effects employed in major feature films. We are rapidly approaching the point, where entire films are rendered and animated using computers without compromising the visual realism, as is showcased in recent epic efforts such as "300" and "Avatar ". The successful application of computers to achieve an unprecedented amount of realism in these and other projects, is in large due to the advances of computational techniques for doing physically based modeling and animation. Especially since films such as "The Day After Tomorrow" paved the way for spectacular fluid animations, we have seen how far the combined talents of researchers, developers and artists can go. It should be stressed, that while the results achievable with the help of these methods often appear physically plausible, they are *not* merely visualizations of actual physical simulations. The techniques strive for realistic and directable appearance rather than physically accurate behavior, and they do so through approximations of the underlying complex physics. This typically leads to improved computational complexity while at the same time retaining the objective — to render the appearance of reality rather than faithfully simulating it. This also means that specialized methods have been and are continually being developed, and there are still many challenging problems which must be solved before physically based methods and fluid simulations in particular can reach their full potential. One particular challenge is that some techniques only work in laboratory settings while others are too hard to control in order to achieve a desired artistic vision. Another avenue which requires even more efficiency and poses stricter constraints on time- and memory-requirements is the rapidly evolving gaming industry. It continues to challenge contemporary techniques as the demand for more realistic games becomes ubiquitous, and in many cases

Figure 1.1 Screenshots of level set and fluid simulations utilized in major film productions. **Top:** The Sandman in "Spider-Man 3". ©Columbia Pictures. **Bottom left:** Chaotic stream in "Ratatouille". ©Pixar Animation Studios. **Bottom right:** The giant maelstrom in "Pirates of the Caribbean 3". ©Walt Disney Pictures.

entirely new mathematical and algorithmic approaches to the problems posed have to be taken.

The level set method, which was originally developed for capturing the propagation of curvature-driven fronts in computational physics [95], is an important example of how well-established techniques from other sciences have found applications in computer graphics and similar areas. Technically speaking, level sets are dynamic implicit surfaces coupled with partial differential equations that govern their movement. These partial differential equations are referred to as the level set equations. Level sets are in many ways ideal for physically based simulations of dynamic surfaces such as free surface fluids, since they handle both topological and geometrical changes in a relatively simple way compared to traditional triangle-based representations. Another important example is the adaptation of *Computational Fluid Dynamics (CFD)* to produce the movement of natural phenomena such as water, smoke and fire. These methods are often coupled with level sets where they produce the velocity field which drives the level set that represents *e.g.* the water surface [30, 88].

Figure 1.1 shows various applications of level sets and fluid dynamics in recent film productions. In "Spider-Man 3" level sets were used extensively to portray the Sandman as shown in the top row. This included complex erosions of surfaces, geometry blending, and efficient collision detection with millions of particles [3, 103], which alone demonstrates the versatility of level sets. The bottom row of figure 1.1 highlights two important aspects of fluid simulations in production, both of which are problematic with respect to using physical models. Firstly, the level of detail is very high. Splashes, foam and mist are small-scale features, even in a relatively confined simulation domain such as the one used in the leftmost scene from "Ratatouille". Often these features are added manually during post-processing by an army of talented artists using the result of the underlying simulation strictly as a coarse basis. This is done simply because the simulation cannot be run at a resolution that is high enough to

capture the desired details. When the scope is as grand as in the rightmost scene from "Pirates of the Caribbean 3", the amount of post-processing work needed is only further increased. In the original level set method [95], a level set surface, or *interface*, is in fact embedded in a volume which is sampled on a dense, uniform three-dimensional *grid*. Both storage and computational requirements scale as N^3 in three dimensions, where N is the number of grid points in each dimension of the grid. This is the reason for the limitations of the physical simulation, since this complexity imposes a strict barrier on the feasibility of the simulations in terms of both memory and computational requirements. Preferably, the requirements should scale with the area of the surface itself. Most productions solve the computational resource aspect by applying narrow band methods [101] which only solve the level set equations near the surface instead of treating the entire grid. The entire grid is still represented, though. The memory aspect has also received a lot of attention with sparse approaches such as octrees, but there has always been an incessant trade-off between efficient storage and fast computational times [11, 72]. Recently, the work of Nielsen and Museth [90] has remedied both aspects of the problem by introducing the DT-Grid sparse data structure with accompanying algorithms, which both scales with the size of the interface rather than the enclosing grid and allows fast sequential access. I have leveraged on this work in some of the research presented in this dissertation, as will be explained in section 1.3.

The second important aspect of production-scale fluid simulations is control. Currently, it is very hard to control and modify animations produced from physical models to obtain a desired behavior or appearance of the fluid. The problem is twofold: Firstly, these models employ numerous non-intuitive parameters that often require trial and error, and secondly, they are very sensitive to the actual resolution of the simulations. This leads to a frustrating situation where an animator will explore the parameter space at a low resolution to maintain fast turnaround times but then realizes that the appearance of the animation changes completely as the resolution is increased to add more details. Concrete examples of this nuisance were encountered in the production of "Ratatouille":

> *We did a coarse low-resolution simulation of the entire length of the rapids for several hundred frames of animation time, and the layout department selected camera angles and animated camera moves based on that* [*simulation*]. [...] *the intent was to simply increase the detail and quality of the simulation, while reproducing the overall motion of the coarse simulation. In practice this proved to be impossible.*
>
> — Eric Froemling, Tolga Goktekin and Darwyn Peachey [36]

The artists were instead forced to "cheat" by inserting invisible obstacles to direct the fluid's overall motion and change the timing. Moreover, often the animator does not want the physically correct solution or behavior, but would rather obtain a certain look or characteristic. In the flooding of New York in "The Day After Tomorrow", cumbersome modifications were employed in a repetitive, trial-and-error fashion in order to prevent a certain building from being flooded. In "300" there is another example of a gigantic simulation of a raving sea, where a very specific look from the comic,

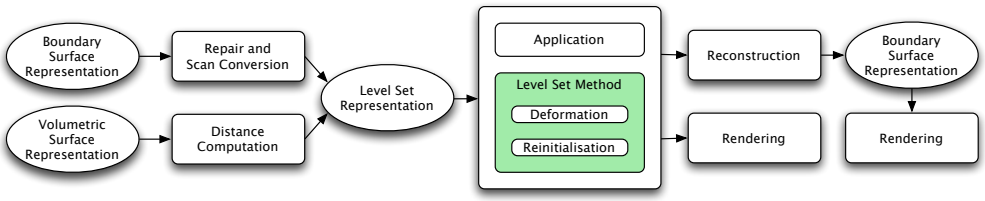


Figure 1.2 The typical level set pipeline. Part of the contributions of this dissertation fall into the area highlighted in green.

on which the film is based, had to be adhered to. A final example of a fluid simulation required to behave in a completely non-physical manner is the tar monster from "Scooby Doo 2: Monsters Unleashed" [133]. Here, the fluid simulation is coupled to an animated character to create a living, breathing blob of tar.

Altogether the issues identified above impose limitations on the scale and amount of detail achievable with existing methods. This is a problem since there is a clear ongoing trend in scientific computing, computer vision, computer graphics and visual effects of increasing simulation resolutions. For the engineering fields an improved numerical accuracy is desired whereas the graphics researchers and artists desire an improved visual accuracy. The epic scales and enormous amount of details present in today's fluid simulations for film productions must be topped by the next generation, while the deadlines and schedules stay the same. While more efficient algorithms and hardware will get us some of the way, facilitating more efficient workflows could be another way of making large simulations more feasible within visual effects. The contributions of this dissertation present and explore new methods for controlling fluid simulations in order to improve the workflow of a visual effects artist.

As mentioned, higher resolutions are also in high demand in scientific computing. Within this field, the time limits are often less strict, and researchers can afford to spend a long time on running *e.g.* level set simulations at high resolution. When these simulations become too big to fit in main memory even when using sparse data structures, special algorithms are required to deal with reading and writing the data to disk. The contributions of this dissertation address this need by developing an efficient framework for streaming level set computations to and from disk.

1.1 Level Set Pipeline

Above we have highlighted some of the applications as well as some of the limitations of physically based animation, in particular for level sets and fluids. Before we proceed to describe our work and contributions, we will provide a brief overview of the typical level set pipeline to give the reader an intuitive understanding of the workflow associated with using this representation, see figure 1.2. One contribution of this dissertation falls into the area highlighted in green in the figure: *The level set method*. This area includes the algorithms that perform the dynamic deformation of the level set surface, and we will return to them shortly. At the leftmost part of figure 1.2, we

see the most common surface representations in which almost all models and data are represented. The *boundary surface representation*, also known as an explicit surface representation, is the prevalent surface representation within computer graphics and is what most artists use to store three-dimensional models. It typically takes the form of triangular meshes, NURBS or subdivision surfaces. *Volumetric surface representations* are becoming quite common as medical and clinical research has started to make use of models produced from CT or MRI scans. The *level set representation* — the data structure used when performing level set deformations — is typically a dense uniform grid (combined with a narrow band data structure), an octree grid or a sparse non-tree based data structure (such as the DT-Grid by Nielsen and Museth [90]). The level set surface is traditionally embedded in a volumetric *signed distance function* since it offers numerical robustness. Very briefly, the signed distance function represents a surface by storing at each point the shortest distance to the surface, multiplied by -1 if the point is inside the surface. Level sets are closed, non-self-intersecting surfaces, while the models created by artists usually contain holes and overlapping triangles. This means that level sets are always physically plausible whereas boundary representations are not. Therefore these models often need to be repaired before they can be scan converted to a signed distance field. Likewise, volumetric data must also be converted to a signed distance field in order to be amenable for level set simulation.

Once the model is represented as a signed distance field, an *application*, such as a fluid simulation or a shape metamorphosis, can utilize the level set representation to deform the surface by means of the *level set method*. The method consists of two logically separate parts. The *deformation* module *advects* or *propagates* the level set, *e.g.* through an externally given velocity field or depending on differential properties of the surface itself, while the *reinitialization* module reinitializes the deformed level set representation to restore its signed distance field property. Depending on the representation employed, other steps might be required. If a narrow band method is used, for example, the narrow band data structure which follows the dynamic surface must be rebuilt as the level set deforms (more on this in chapter 3). The resulting surface of the level set simulation can either be *rendered* directly using a ray tracer [111], or converted back to a boundary surface representation using a standard *reconstruction* method such as the marching cubes algorithm [71]. It could also be used as the input to another application, such as a fluid simulation.

1.2 Fluid Simulation Pipeline

Faithfully reproducing fluid phenomena for visual effects is a challenging computational task with several, often conflicting goals such as producing a visually satisfying, highly detailed result while retaining a controllable expression. In order to appreciate how hard this can be, we will also briefly give an overview of the typical fluid simulation pipeline. Figure 1.3 shows a basic view of the typical pipeline for simulating smoke and water. This particular pipeline utilizes a numerical scheme for solving the *incompressible Navier-Stokes equations* which is referred to as the *Stable Fluids* solver [114]. We will explain the equations in detail — including how they govern the motion of fluids — in chapter 5, and for now give an overview of how they are solved

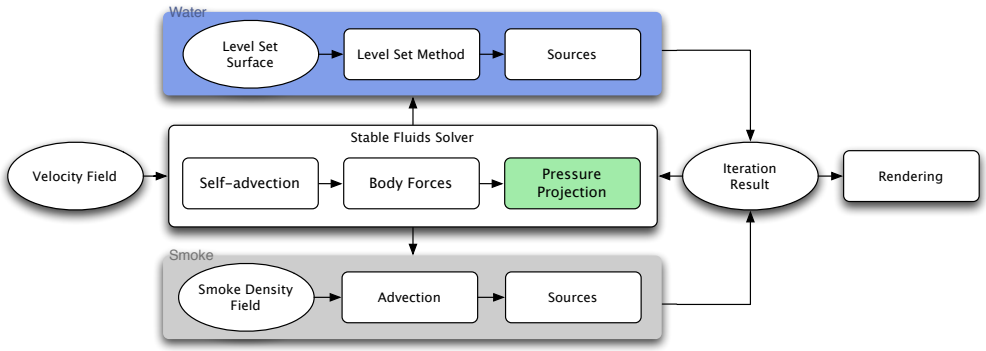


Figure 1.3 The typical fluid simulation pipeline. The same numerical solver is typically used at the heart of both smoke and water simulations. Essentially this entire figure could be fitted within the *application* box of figure 1.2 as level sets are typically used for representing solid boundaries and water surfaces. Part of the contributions of this dissertation fall into the area highlighted in green.

in practice.

We are simulating the motion of the fluid, which at any given time is represented by a velocity field, and the task is thus to integrate this field over time. The Stable Fluids solver consists of three modules that are applied in succession to perform this integration: A *self-advection* module advects the velocity field through itself. Next, *body forces*, *i.e.* forces that are applied in the entire fluid, accelerate the velocity field. Examples of body forces are gravity and buoyancy. After these two modules have been applied the velocity field will likely be diverging or compressible, *i.e.* the volume of the fluid is changing, and a *pressure projection* module is needed to ensure that the velocity field becomes incompressible again. This is a very important step as simulating compressible fluids is very expensive and often unnecessary since the compressibility of most fluids of interest in computer graphics is negligible[1] [12]. There is sometimes an extra step involved in the Stable Fluids solver called *diffusion*. It is usually placed just before the pressure projection module. It is required to simulate viscous fluids such as honey or when simulating very small-scale fluid flows. However, many fluids we are interested in animating in computer graphics, such as water or air, are largely unaffected by viscosity. Such fluids are called *inviscid*. We will see in chapter 2 that any numerical method actually introduces some amount of *numerical viscosity* or *numerical dissipation* which can dampen the liveliness of a simulation [12].

However, just having an evolving velocity field will not give us any visible behavior. As indicated by the top and bottom rows of figure 1.3, the pipeline for simulating smoke and water are usually slightly different. For smoke, we can *advect* soot particle densities (*i.e.* smoke densities) sampled on a grid through the velocity field from the Stable Fluids solver. New smoke is added at *sources*. Note that other quantities,

[1] One example of an exception is explosions which cause shock waves that compress the fluid. However, from a visual viewpoint, they are nearly invisible and move extremely fast, and most audiences have no idea how they behave. Therefore it seems like a better idea artistically to create something that works visually rather than attempting to simulate them accurately.

such as air temperature, can be integrated in the exact same manner. Also, note that the various quantities can be used as input to compute for instance body forces in the Stable Fluids solver, such as buoyancy resulting from temperature changes. This is indicated with the arrow from the *iteration result* back to the Stable Fluids solver. For water, a level set is typically used to represent and visualize the water surface. It is also used to define a boundary condition on the Stable Fluids solver since we only integrate velocities inside the water. The level set method is utilized to advect the surface using the velocity field. The advected surface is then fed back into the Stable Fluids solver in order to update the boundary conditions.

1.3 CONTRIBUTIONS

The overall goal for this dissertation has been to improve the management of large computations within scientific computing, computer graphics and physically based animation. I have approached this through a couple of focus points which can be formulated as the following research objectives:

- To improve the feasibility of using level sets at high resolutions in computer graphics and scientific computing applications.
- To improve artistic control and efficiency in the typical workflow surrounding fluid simulations in visual effects.

I have been a major contributor to the design and implementation of algorithms and models within both research projects (as documented by the co-author statements) , and I have enjoyed constructive and helpful discussions with my co-authors. Therefore I will be using the pronouns *we* and *our* when referring to the work presented in this dissertation. Below follows an overview of the contributions of the entire dissertation, divided into two distinct categories corresponding to the research objectives.

OUT-OF-CORE LEVEL SET STREAMING FRAMEWORK

We propose an external memory framework for efficient streaming of level set computations. *External memory* or *out-of-core* algorithms utilize external memory, such as disks, as a way of overcoming the limited internal (*in-core*) physical memory during execution. In particular, the contributions include code transformations of the level set algorithms in order to provably maximize locality of memory references. The transformations allow us to reduce the number of passes over the data during computations. Specifically, we only have to stream data once for each iteration of the level set computation, which consists of an advection, a reinitialization and a narrow band rebuild step. For some level set simulations, we can even combine as many iterations into one pass as allowed by the physical memory. Our transformations include a tiling which enables computation on each tile independently and hence in parallel. The framework is built upon the DT-Grid data structure and the out-of-core framework of Nielsen *et al.* [91] which gives us a near optimal page-replacement algorithm and prefetching strategy

in the context of sequentially accessing out-of-core level sets with finite differences[2]. The contributions of our new framework result in computations that are CPU limited rather than *Input/Output (I/O)* limited, *i.e.* they are not stalled by I/O operations and the CPUs are working at full capacity. This is because each I/O operation is amortized over more CPU cycles. Extensive benchmarks show that our new framework sustains a throughput of 77% – 92% of an internal memory simulation independent of the resolution of the level set. It is also demonstrated that the throughput is sustained when several simulations are run on the same disk, and that the parallel computations exhibit near-optimal scaling as the number of CPU cores are increased. Figure 1.4 shows an example of a high-resolution simulation (2048^3) using our out-of-core framework.

Fluid Simulation Control Framework

We present a framework for controlling high-resolution fluid simulations using low-resolution prototype simulations. The central idea is to support the typical workflow of a visual effects artist in which a coarse prototype simulation is used for tuning parameters and simulation setup. When the coarse setup gives satisfactory results, additional details are added by increasing the simulation resolution. However, in many cases increasing the resolution completely changes the behavior of the simulation in addition to adding the desired high-frequency detail. Our framework couples the velocity field of a prototype to the low-frequency flow (*i.e.* the low frequencies of the high-resolution velocity field) of a high-resolution simulation. We say that the low-resolution simulation is *guiding* the high-resolution simulation. The assumption is that this ensures an overall correspondence in the bulk movement of the high-resolution flow while allowing new, dynamic high-frequency details to develop. Our main contributions are two mathematical models for expressing this coupling based on *calculus of variations* (see [63] for a good introduction). The models result in a modified pressure projection step while the rest of the fluid simulation pipeline remains unchanged. We have implemented a customized memory-efficient, parallel multigrid solver in order to be able to solve the resulting linear systems efficiently. We demonstrate the framework by coupling several smoke simulations and exploring various artistic effects obtainable by adjusting where and how tight the guiding should be. Figure 1.5 shows the results obtainable by using our fluid control framework. Notice the high correspondence between the coarse, low-resolution prototype at the top and the high-resolution result at the bottom.

1.4 Outline

This dissertation consists of two parts: *Part I: Overview* which provides a coherent presentation of the research done in this PhD project as well as the background material needed to appreciate the methods employed. *Part II: Papers* collects the publications that form the result of our work. As such, part II constitutes the more technical portion

[2] A finite difference is a robust numerical method for approximating derivatives. An elaborate explanation will be given in chapter 2.

of this dissertation, while part I serves as a comprehensive introduction to the material presented in the papers. The content of the individual chapters and papers is as follows.

Part I: Chapter 2 introduces implicit surfaces as well as the basic level set theory, including the original level set method. It also presents a number of numerical schemes for solving the level set equations. Chapter 3 provides an overview of previous and concurrent extensions and improvements to the level set method relevant for the work in this dissertation. Next, chapter 4 describes our out-of-core framework for streaming computations on high-resolution level sets, and it supplements paper I by more thoroughly introducing memory hierarchies and code transformations as well as providing further evaluations.

Having described the first major contribution of this dissertation in chapter 4 and paper I, chapter 5 changes focus and motivates fluid simulations in the context of computer graphics and visual effects. It furthermore introduces a basic method for simulating incompressible fluids by solving the Navier-Stokes equations numerically. Subsequently, chapter 6 gives an overview of previous and contemporary improvements to this basic numerical method. Chapter 7 then motivates the need for methods for controlling fluid simulations. It briefly introduces calculus of variations and explains how it can be used in the context of fluid simulations. A discussion of a number of other control methodologies that complement and can be used in conjunction with our control framework presented in papers II and III is also provided. In addition, it presents further implementation details of the framework which will form part of an upcoming journal paper.

Finally, chapter 8 discusses future work and concludes the dissertation.

Part II: Paper I is a journal paper which describes the out-of-core framework for streaming computations on level sets of high resolution. It provides code transformations for the level set algorithms, such as advection, reinitialization and rebuilding of the narrow band, which allows us to combine several passes over the data into one. It also introduces the *Tiled DT-Grid* data structure which allows for simultaneous computations on separate tiles. Furthermore, we show that the transformations maximize data locality. We demonstrate that these transformations result in parallelizable, CPU limited algorithms that retain a steady throughput independent of disk latency.

In paper II we present the first iteration of our fluid control framework. We develop a mathematical model based on calculus of variations for ensuring correspondence between a low-resolution prototype simulation and the low-frequency flow of a high-resolution simulation. *Guiding weights* are introduced to specify the strength of this coupling. We implement a customized, parallel multigrid solver for the resulting system of linear equations and derive a discretization of the penalization method for handling boundary conditions. Finally, it is demonstrated that the method can be used to couple high-resolution simulations to velocity fields obtained through physical simulation as well as artistically created velocity fields.

Paper III improves upon and somewhat subsumes the work in paper II. In it the mathematical model is improved by leaving the high-frequency components of the

high-resolution simulation completely out of the coupling with the low-resolution prototype simulation. This results in a model where the matrix does not have to be recomputed for each iteration when the guiding weights change over time and effectively makes time-dependent guiding weights feasible. We demonstrate that time-dependent guiding allows for more high-frequency detail to develop, and we explore various criteria for choosing the guiding weights.

Figure 1.4 Level set advection of eight spheres in a divergence-free and periodically symmetric velocity field. The resolution is 2048^3. Due to the properties of the velocity field, the eight spheres should return to their original shape after one period. This is illustrated in the bottom row of pictures. The deformation results in very thin features which require a high resolution in order to be resolved properly. If the resolution is not high enough the spheres will not return to their original shapes.

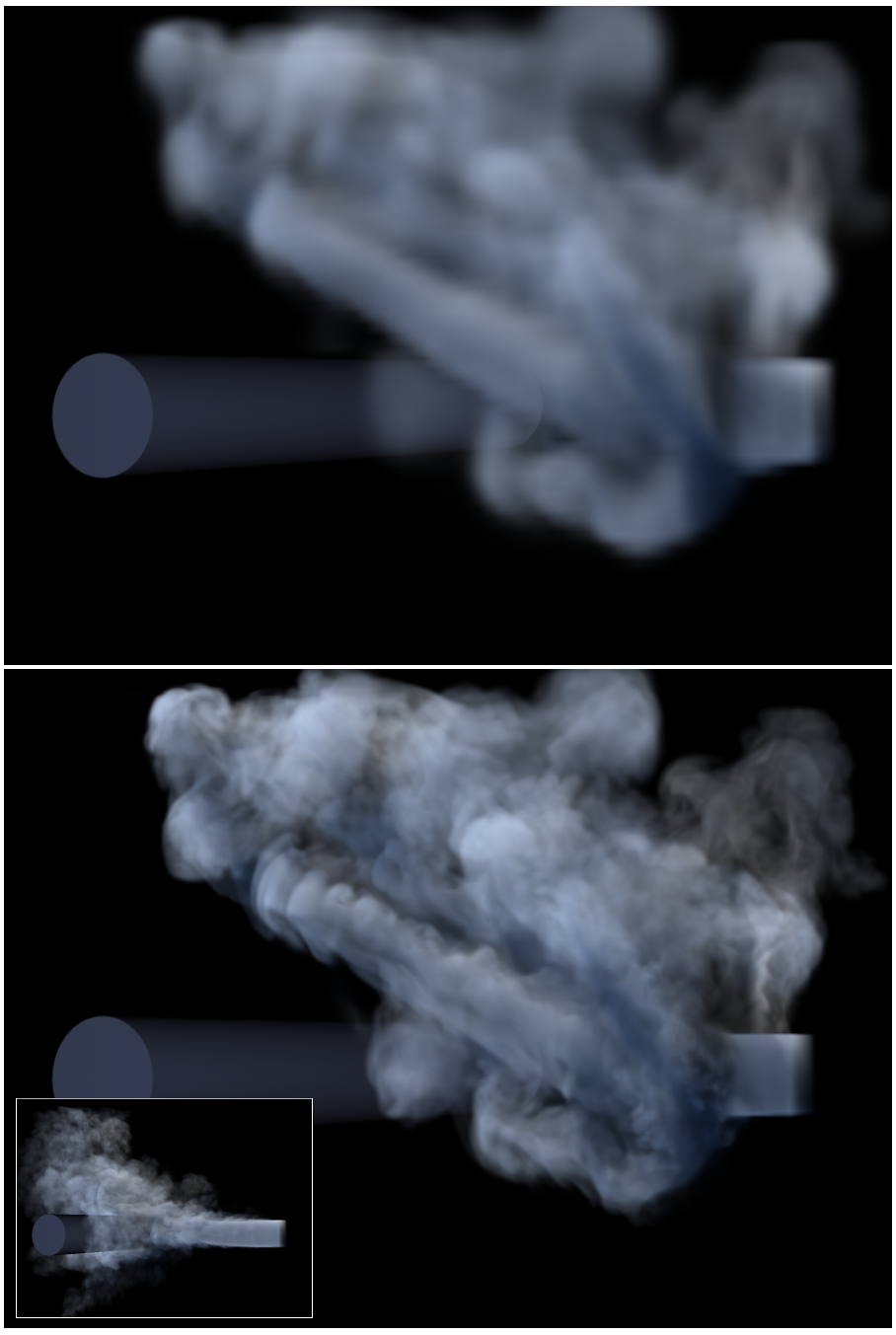


Figure 1.5 **Top:** Smoke simulation prototype in which hot smoke is injected into the air and glides over the column. The prototype represents the desired result but is too coarse and lacking appropriate details. **Bottom:** High-resolution simulation which has been coupled with the prototype to ensure resemblance while at the same time adding dynamic high-frequency detail. **Insert:** Illustrates a normal, uncoupled high-resolution simulation which clearly does not correspond very well with the prototype since the column is enveloped in smoke.

PART I

OVERVIEW

CHAPTER 2

LEVEL SET METHODS

In our everyday life we constantly interact with complex and beautiful surfaces and deformations. One of the best examples is our interaction with water which is so natural to us that we hardly ever acknowledge or treasure the intricate ways in which it merges, forms and breaks up. It is to a large degree the ambition of computer graphics to simulate and reproduce the appearance and dynamic behavior of phenomena in the world around us. Simulation is particularly useful when the effect of a certain phenomenon is not easily obtainable through the skills of an artist or animator. In order to lift this task, computer graphics unites several different scientific disciplines. For instance, in order to arrive at computational techniques capable of reproducing a realistic behavior of water, one needs to resolve to theory and practice from the disciplines of mathematics and physics. Many ideas are also retooled from fields such as engineering. A *level set* is a mathematical construction which captures a *dynamic implicit* surface that possesses the properties required to represent complex surface deformations such as those water undergoes. The adjective *dynamic* refers to the ability of the surface to change over time, while *implicit* refers to how the surface is represented. The level set method was introduced by Osher and Sethian [95] in 1988 as a method for tracking interfaces (*i.e.* surfaces) in computational physics. Since then considerable efforts have been and still are put into the development of more accurate and robust numerical methods for solving the accompanying equations which govern the dynamic behavior of a level set. Level sets have also found use as a popular surface representation in several other problem areas spanning multiple fields. In computer graphics and vision in particular these areas include (but are certainly not limited to) fluid simulations of water and fire [24, 30, 88], collision detection in particle simulations [3, 103], geometric modeling [83], shape metamorphosis [10], and segmentation of volumetric data sets [64, 131]. Examples of the applications mentioned above can be found in figures 2.1 and 2.2. Further examples of level set applications in image processing, computer vision, and computational physics can be found in the book by Osher and Paragios [94] to which we refer the interested reader.

Level sets provide a number of unique advantages compared to many other surface representations. A level set cannot self-intersect, *i.e.* the surface cannot cross over it-

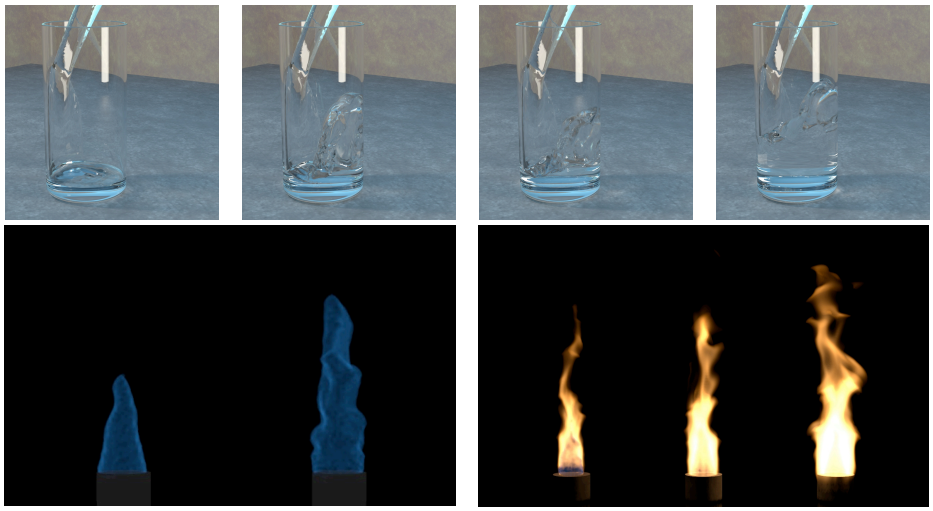


Figure 2.1 **Top:** Even a mundane act such as pouring a glass of water results in a highly complex and detailed behavior. The water surface is represented by a level set. Reprinted from [24]. **Bottom:** A physically based simulation of fire. The blue core of the flames is represented using a level set. Reprinted from [88].

self. As we will see in the next section, this is a simple consequence of the definition of level sets. Furthermore, complex topological changes are handled automatically by the underlying mathematics. These properties are not shared by explicit representations, such as triangle meshes, which are the most widely used surface representation within computer graphics today. Finally, the numerical schemes for the dynamics of level sets are in many ways relatively simple to employ. These unique features are all utilized by the applications mentioned above. For example, fluid surfaces such as water often undergo complex topological and geometrical changes as they merge and pinch off in elaborate ways. These changes are handled automatically by employing level sets as the underlying surface representation [30] (see figure 2.1). In shape metamorphosis, topological changes are handled gracefully as well when utilizing level sets (see figure 2.2). When doing geometric modeling with traditional surface representations, such as triangle meshes, one often ends up with self-intersecting geometry due to the inherent properties of the common work flows. If the final models are to be used for physically based simulation or physical prototyping this is undesirable. Level sets can be used in the modeling process to avoid this issue. Also, various surface editing operators that are difficult to perform with triangle meshes can be easily utilized with level sets [83]. These surface editing operators can be used to repair digital models scanned from real world geometry. Using physical models when creating for instance characters is a very popular technique in many feature film productions, and therefore the ability to repair the scanned geometry is very useful.

The dynamic nature of the level set rests on a solid and rather advanced mathematical and numerical framework which elegantly generalizes to any dimension. We are, however, first and foremost concerned with computer graphics and thus mostly work in three dimensions. Therefore in this and the following chapters we will restrict ourselves to descriptions in three dimensions and further reduce to one or two dimen-

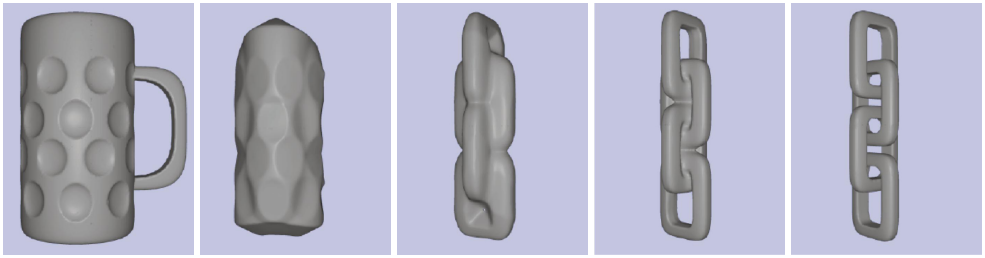


Figure 2.2 An example of shape metamorphosis in which a beer mug morphs into a four-link chain. The level set gracefully handles topological changes during the metamorphosis. Reprinted from [10].

sions when it serves the exposition. This chapter will focus on describing the basics of the level set method while the next chapter will look at extensions and improvements leading up to some of the contributions presented in this dissertation. In the next section we will briefly describe the ideas behind implicit surfaces and contrast them to the explicit surface representation. We will then discuss the theory behind the dynamics of the level set method and look closer at the equations which govern the movement. Finally, we describe the numerical schemes which implement the level set theory on a computer.

2.1 Implicit Surfaces

Implicit surfaces and their properties are generally well understood mathematically, and they come in many distinct forms within computer graphics, each associated with its own theory and unique properties. An *explicit* surface representation explicitly specifies the points on the surface. Expressed mathematically, an explicit surface representation provides a map between a parameter space and the points on the surface. Within computer graphics, there are several examples of explicit surfaces. They are typically *sampled* (*i.e. discrete*) representations in the sense that they specify a finite number of points possibly along with information on connectivity and how to interpolate the surface between them. Triangle meshes, point-based representations, NURBS, and subdivision surfaces are all examples of such explicit surface representations.

An *implicit* surface representation instead specifies a surface as the *isocontour* of a scalar function. Mathematically, given a scalar *embedding function* $\phi : \mathbb{R}^3 \to \mathbb{R}$, an implicit surface is represented as the preimage, $\phi^{-1}(k)$, of some scalar k. In other words, the surface consists of the set of points $\mathbf{x}$ in $\mathbb{R}^3$ for which $\phi(\mathbf{x}) = k$. Usually, and without loss of generality, we restrict ourselves to look at the *zero isocontour* where $k = 0$. Since a two-dimensional surface is defined by a three-dimensional embedding function, we say that the implicit surface has co-dimension one.

Spheres and circles are shapes which are easily described implicitly. A circle of radius r is for example given by the expression $\phi(x,y) = \sqrt{x^2+y^2} - r = 0$. The left side of figure 2.3 shows this example, while an explicit representation of the same circle is given to the right. Note how an implicit surface representation does not directly

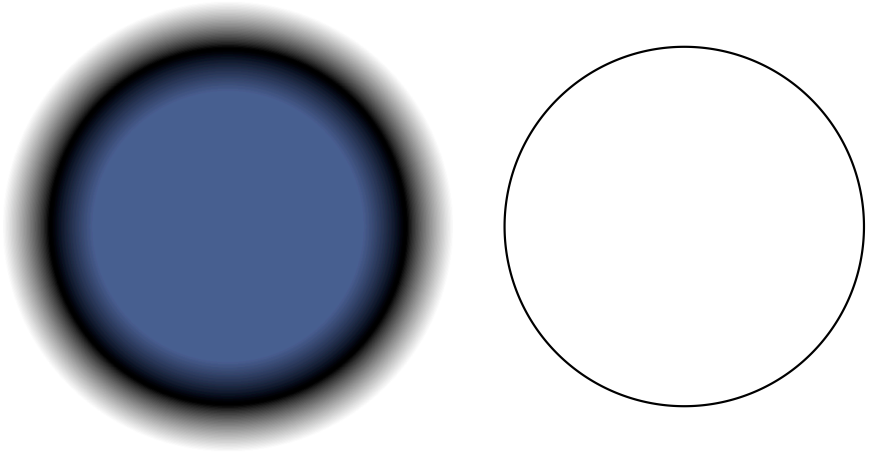


Figure 2.3 Left: Circle implicitly represented as the zero isocontour of the embedding function $\phi(x,y) = \sqrt{x^2+y^2} - r$. The color has been clamped in order to get a noticeable color gradient near the interface. **Right:** Circle explicitly represented as $(\cos(\theta), \sin(\theta))$ with $\theta \in [0; 2\pi)$.

specify the points on the surface. Instead it allows you to query whether or not a given point lies on the surface. Up front, this may seem to make implicit representations inferior to explicit ones, and there are some applications within computer graphics where this is indeed the truth. As we will see in a moment, however, very powerful tools are readily available when using the implicit representation.

The level set method only deals with closed implicit surfaces. This means that the surface must partition $\mathbb{R}^3$ into clearly defined interior and exterior regions, denoted Ω^- and Ω^+, respectively. We will assume for the rest of this dissertation that the implicit surface is given by the zero isocontour, and that the embedding function maps points in the interior region to negative values, while points in the exterior region are assigned positive values. This sign convention can be seen to the left in figure 2.3 where the blue color is associated with the interior region of negative values while the exterior region of positive values is shown in white.

At this point, we are able to point out two important advantages of level sets which follow from the above properties. Firstly, for a given point in space we can determine whether it is inside or outside the surface simply by evaluating the embedding function at that point and looking at the sign. For explicit representations, such as meshes, this kind of query is more complicated [5] and the result is ambiguous if the mesh contains holes or self-intersections. Secondly, a level set cannot contain self-intersections. This stems from the simple fact that an implicit surface is represented by a *single-valued* embedding function. Thus any point in $\mathbb{R}^3$ cannot both have a negative and a positive sign at the same time. As we will see in chapter 5, this property is very important for water simulations which require distinctly separable regions of space. Meshes on the other hand can easily become self-intersecting when deformed over time.

Often we do not represent implicit surfaces analytically, simply because no analytical expression is available or known for a given surface. Instead the embedding

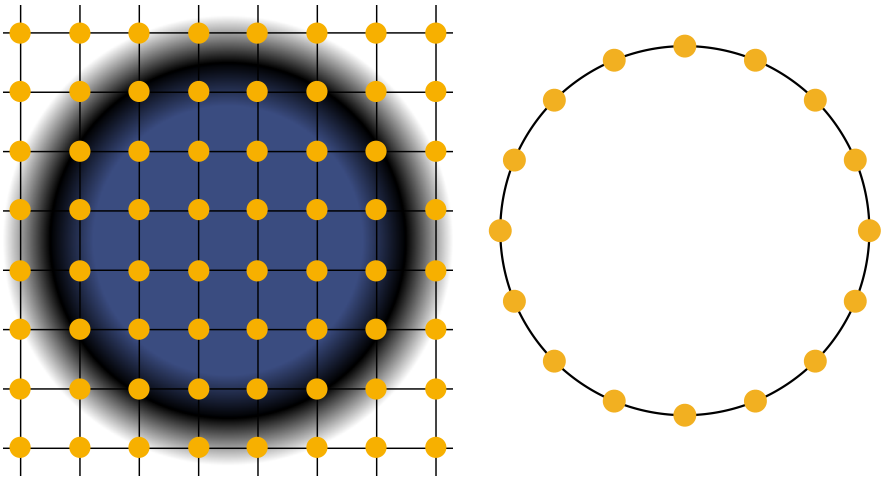


Figure 2.4 **Left:** Circle implicitly represented as the zero isocontour of the embedding function $\phi(x,y) = \sqrt{x^2+y^2} - r$ sampled on a dense uniform grid. The rather coarse sampling serves to illustrate the principle of an Eulerian representation. **Right:** Circle explicitly represented by points connected by line segments. Again a rather coarse sampling has been used to illustrate the principle of a Lagrangian representation.

function is sampled on a *grid* as shown to the left in figure 2.4. A sampling of an explicit representation is shown at the right side of the same figure. Sampling an implicit representation on a fixed grid is referred to as an *Eulerian* representation because it captures the interface rather than tracking it. Sampled explicit representations are on the other hand often referred to as *Lagrangian* representations. Notice that grid points in an Eulerian representation remain fixed during deformations. It is the changes to the scalar values of the sampled embedding function that cause the surface to move. In a purely Lagrangian representation it is the sample points that move throughout deformations. Various kinds of grids have been suggested, and the dense uniform grid of figure 2.4 is but one of the most common. We will return to this issue in chapter 3.

Placing the implicit surface in an embedding function gives access to a powerful differential toolbox. In particular the surface normal pointing outwards can be computed directly from ϕ as the normalized gradient

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|} \tag{2.1}$$

while the mean curvature in three dimensions is given by[1]

$$\kappa = \frac{1}{2}\nabla\cdot\vec{N} \tag{2.2}$$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ is a differential operator. Other quantities such as surface and volume integrals can be computed in a similar simple manner as explained in the book by Osher and Fedkiw [98].

[1] The factor of $\frac{1}{2}$ is usually omitted in the literature. Please refer to [84] for a correct derivation.

From a theoretical viewpoint, the embedding function is immaterial as long as it is Lipschitz continuous. Even so, one particular class of functions has proven itself very useful in computer graphics and level set simulations. The *signed distance function* assigns to each point in $\mathbb{R}^3$ the shortest distance to the surface. This distance is multiplied by -1 for points in the interior region. The expression given previously for the implicit representation of a circle is in fact a signed distance function. Many operations and formulas simplify as a result of the properties of the signed distance function. Besides being able to distinguish if a point is in the interior or exterior region using a single lookup, we are now also given the shortest distance to the surface. Such information can be useful when rendering using ray tracing since it enables a technique known as ray leaping [24]. Also, when simulating for instance rigid bodies or cloth, it simplifies collision detection and response since any point $\mathbf{x}$ in $\mathbb{R}^3$ can easily be projected unto the closest point on the surface $\mathbf{x}_s$ using the formula $\mathbf{x}_s = \mathbf{x} - \phi(\mathbf{x})\nabla\phi(\mathbf{x})$ [13, 39]. Furthermore, the length of the gradient of a signed distance function is identically one, *i.e.* $|\nabla\phi| = 1$, except at corners or on the medial axis[2] where the gradient is not defined. As $|\nabla\phi| = 1$, the formulas for the surface normal and the mean curvature, simplifies to the gradient ($\vec{N} = \nabla\phi$) and the Laplacian ($\kappa = \frac{1}{2}\Delta\phi$), respectively. For explicit representations such as triangle meshes, the computation of differential properties or location of the closest point on the surface is not as simple [5].

2.2 The Level Set Method in Theory

So far we have established how a level set is *represented*. Specifically, we will represent a level set implicitly using an embedding scalar function ϕ which we will require to be a signed distance function. As mentioned previously, this type of function is Lipschitz continuous and smooth except at kinks which makes it well-suited for numerical simulation. We will now turn our attention to the theory of the level set method [95] which adds *dynamics* to implicit surfaces. In the next section we will consider how the equations presented in the following can be discretized on a computer and briefly describe the numerical methods developed for solving them.

2.2.1 The Level Set Equations

The dynamics of a level set are manifested in a *level set equation* which is to be solved in order to make the interface move. Level set equations exist in many different variations and depending on the problem or application at hand we should pick the best suited one. We will start by considering the most widely used form within physically based simulations such as water simulation. In this case the motion of the surface is given by a time-dependent velocity field $\vec{V}(\mathbf{x},t) : \mathbb{R}^4 \rightarrow \mathbb{R}^3$ which for each point in space at a given time assigns a vector describing the velocity of the embedding function, *i.e.* the surface. The velocity field is obtained for each time step by solving the Navier-Stokes equations for fluid flow. Now consider a point $\mathbf{x}$ lying on an implicit

[2] The medial axis is the set of points which are equidistant to several points on the surface, *e.g.* the center of a sphere

surface which moves over time. The movement of the point will trace out a path which is given by the expression $\mathbf{x}(t) = (x(t), y(t), z(t))$. If we couple this expression with the embedding function, ϕ, thereby adding dynamics to it, we get the following time-dependent equation for the surface given by the zero isocontour: $\phi(\mathbf{x}(t), t) = 0$. Differentiating with respect to t and using the chain rule, we obtain the equation

$$\frac{\partial \phi}{\partial t} + \frac{d\mathbf{x}}{dt} \cdot \nabla \phi = 0 \tag{2.3}$$

We immediately see that $\vec{V}$ can replace $\frac{d\mathbf{x}}{dt}$. Therefore, given a time-dependent velocity field $\vec{V}(\mathbf{x}, t)$, we can solve the equation above to evolve or *advect* the surface forward in time. Equation 2.3 is a *partial differential equation (PDE)* as it involves derivatives with respect to several variables. Equation 2.3 is also an *initial value problem*. Starting from an initial embedding function ϕ, which captures the interface of interest, the PDE is solved at each time step in order to evolve the interface.

By using the relationship between the normal and the gradient of the embedding function from equation 2.1, we can obtain another level set function directly from equation 2.3. To do this, we split the velocity field $\vec{V}$ into vector components $V_N\vec{N}$ and $V_T\vec{T}$ which are normal and tangential to the interface, respectively. V_N and V_T are scalar fields which specify the speed in these two directions. We then get $\vec{V} \cdot \nabla \phi = (V_N\vec{N} + V_T\vec{T}) \cdot \nabla \phi = V_N \frac{|\nabla \phi|^2}{|\nabla \phi|} = V_N |\nabla \phi|$ which changes equation 2.3 into

$$\frac{\partial \phi}{\partial t} + V_N |\nabla \phi| = 0 \tag{2.4}$$

The tangential component disappears and only the normal component of the velocity is important for the movement of the interface. Having these two equations might seem redundant, but they allow us to choose the formulation that is most convenient for a given problem. As mentioned above, when deforming the surface of a water simulation, equation 2.3 is more convenient, since the motion is described by a velocity field obtained from solving the Navier-Stokes equations. Equation 2.4 comes in handy in the context of smoothing a noisy surface, eroding or dilating it, or performing a shape metamorphosis, since these operations are more easily expressed in terms of their speed in the normal direction. In general, the speed in the normal direction is called a *speed function* and it can depend on anything from spatial and temporal position to geometrical and differential properties of ϕ. Smoothing a surface can for instance be achieved simply by setting the speed function to $V_N = -\kappa$, where κ is the mean curvature. We have now looked at two of the most common level set equations within computer graphics. For a number of other examples, please refer to the book by Osher and Fedkiw [98].

2.2.2 Reinitializing the Signed Distance Function

When the embedding signed distance function is subjected to movement caused by any of the above equations it will generally cease to be a signed distance function. This happens because the isocontours do not all move at the same speed, causing

them to bundle together at some points and spread apart at others. Since we want to keep the nice advantages of using a signed distance function, we reset the embedding function to a signed distance function after each advection operation. Note that this operation must *avoid* moving the location of the interface. There are numerous ways of doing this, but in the context of the work presented in this dissertation only two are considered. The first method involves solving the *reinitialization equation* [101, 118]

$$\frac{\partial \phi}{\partial t} + S(\phi)(|\nabla \phi| - 1) = 0 \tag{2.5}$$

where $S(\phi)$ is a *sign function*. A simple example of a sign function is $S(\phi) = \text{sign}(\phi) \in \{-1, 0, +1\}$ which just takes on the sign of ϕ, *i.e.* 1 in Ω^+, -1 in Ω^-, and 0 on the interface. Equation 2.5 must be solved to *steady state* which means that $\frac{\partial \phi}{\partial t} = 0$. This condition further implies that $|\nabla \phi| = 1$ which is what signifies a signed distance function. Notice that equation 2.5 can be viewed as propagating distance information in the normal direction with speed $S(\phi)$.

At the risk of getting ahead of ourselves, we will briefly look at a numerical issue raised by applying the simple sign function given above. It does not work very well numerically as it usually does *not* keep the location of the interface fixed[3]. Instead we will use a smeared out version originally proposed in [101]

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla \phi|^2 (\Delta x)^2}} \tag{2.6}$$

where Δx is the distance between two adjacent sample points in the grid. This smeared out sign function tends to work well in practice since it significantly reduces the tendency to move the surface.

A second method for constructing a signed distance function can be derived by considering the *Eikonal equation*

$$|\nabla \phi| = 1 \tag{2.7}$$

This equation is a non-linear PDE and it has no explicit time dependence. Therefore, rather than being an initial value problem, it is a *boundary value problem*. In a boundary value problem, we are given the values of the function on the boundary of the domain and then solve the PDE to find the remaining values. In this case the boundary is the zero isocontour, and we solve the equation in order to find the signed distance values away from the surface. Although the Eikonal equation is not time-dependent, it can be viewed as propagating the boundary (*i.e.* surface) outwards in the normal direction with the unit speed function. The value obtained at each point is then just the *time of arrival* of the boundary, which equals the distance to the surface due to the unit speed propagation. The next section will describe a couple of fast methods for solving this equation.

We have now briefly touched upon the theory of the level set method. Many issues have been ignored as we have only focused on the details needed to understand the work presented in this dissertation. Again, we encourage the interested reader to consult the book by Osher and Fedkiw [98] for a more thorough overview.

[3] The numerical problems stem from the fact that the function is not band-limited and therefore not possible to represent accurately using any sampling.

2.3 The Level Set Method in Practice

In this section we turn our attention to how the theory from the previous section can be discretized and solved on a computer. However, we will first briefly explain why explicit surface representations are not always well-suited for general surface deformations based on numerical simulations. While deformations of explicit surfaces occur all the time in computer graphics, they primarily work well with minor surface deformations such as most character animations. Large deformations on the other hand may cause the distribution of sample points on the surface to degrade dramatically. If the distance between them becomes too great, aliasing artifacts occur, and if it becomes too small, singularities in differential properties may appear. The latter is particularly disastrous from a numerical simulation viewpoint. Although there are remedies for this such as remeshing, smoothing and similar "interface surgery", these operations are nontrivial and nonphysical, and may therefore affect the simulation in unpredictable and potentially undesirable ways. Level set methods do not suffer from these numerical issues since they utilize a grid of *fixed* sample points. As with many other Eulerian schemes in numerical simulation, they do however inherently introduce an undesirable smoothing of the solution known as *numerical dissipation* or *artificial viscosity*. We will touch upon the origin and consequences of this phenomenon at the end of this chapter.

2.3.1 Finite Difference Approximations of Derivatives

We will now turn to the discretization of differential operators on a uniform Eulerian grid. However, before we move on, let us introduce the notation we will use. Assume that the distance between adjacent points in the grid is given by Δx, Δy and Δz in each dimension respectively. The time step used in the simulation will be given by Δt. The notation ϕ^n_{ijk}, where i, j, k and n are integers, will then be shorthand for $\phi(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$. The temporal and/or spatial dependence may be omitted for clarity when not relevant.

A *finite difference (FD)* [60] is a discrete approximation to a derivative at a given point x based on a Taylor expansion around that point. For instance, a one-sided *forward* FD approximation to $\frac{\partial \phi}{\partial x}$ can be derived from the Taylor series approximation to the point $x+\Delta x$ around x : $\phi(x+\Delta x, y, z) = \phi(x, y, z) + \Delta x \frac{\partial \phi}{\partial x} + O(\Delta x^2)$. By rearranging the terms and dividing by Δx we obtain $\frac{\partial \phi}{\partial x} = \frac{\phi(x+\Delta x, y, z) - \phi(x, y, z)}{\Delta x} + O(\Delta x)$. Ignoring the $O(\Delta x)$ term, whereby we introduce a *truncation error*, we obtain the one-sided forward FD, $\phi^+_x = \frac{\phi_{i+1jk} - \phi_{ijk}}{\Delta x}$. We say that this approximation is *first order accurate* since the order of the truncation error is $O(\Delta x)$. Generally speaking, an FD approximation with a truncation error of $O(\Delta x^n)$ is said to be *nth order accurate*. Similarly, we can derive first order accurate one-sided *backward* and second order accurate *central* approximations to $\frac{\partial \phi}{\partial x}$, obtaining $\phi^-_x = \frac{\phi_{ijk} - \phi_{i-1jk}}{\Delta x}$ and $\phi^o_x = \frac{\phi_{i+1jk} - \phi_{i-1jk}}{2\Delta x}$, respectively. The set of grid points required for the computation of an FD is referred to as the *stencil*. The two-dimensional stencil for the FD approximations presented so far can be seen to the left in figure 2.5.

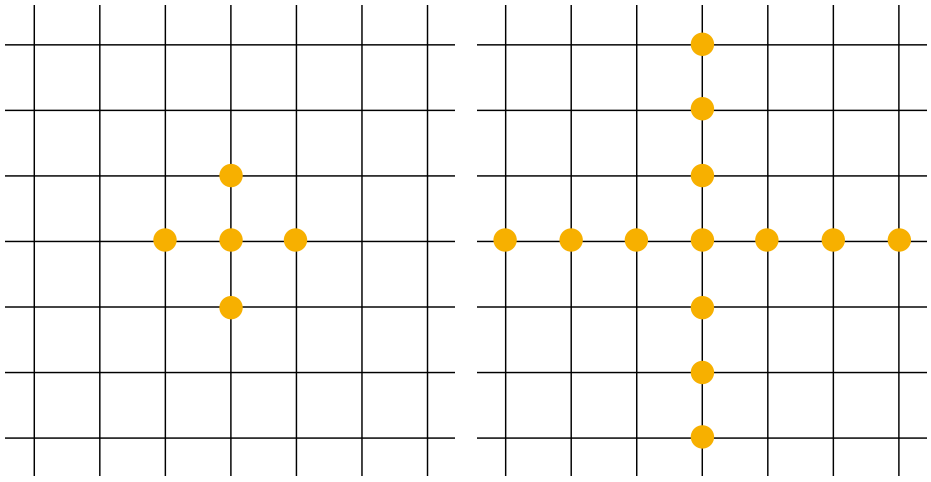


Figure 2.5 Examples of finite difference (FD) stencils. **Left:** The stencil for a first order one-sided forward/backward FD or a second order central FD. **Right:** The stencil for a third order HJ ENO FD or a fifth order HJ WENO FD.

Differential properties such as the surface normal and curvature can easily be approximated by similar FDs. The actual choice of approximation depends on the desired accuracy and the properties of the equation we are solving. Spatial and temporal derivatives are handled differently in practice, and therefore we shall elaborate on them separately. We will start with the spatial derivatives.

Spatial Derivatives

Several higher order schemes for FD approximations have been specifically developed for level sets. They strive to both represent the surface with higher accuracy *and* maintain sharp corners and edges, *i.e.* regions with discontinuous derivatives. The *Hamilton-Jacobi Essentially Non-Oscillatory (HJ ENO)* FD scheme [96] uses Newton polynomial interpolation [60] to reconstruct ϕ as a polynomial which is then differentiated to obtain the approximations. HJ ENO favors the smoothest possible polynomial interpolation of ϕ since this minimizes overshoots in the interpolating function near discontinuities in the derivative. This leads to better approximations near corners and edges, since these overshoots could potentially result in oscillations and numerical errors in the approximation of the derivatives. Third order accurate HJ ENO FD schemes are common in level set simulations, but they can be constructed for any order of accuracy.

The *Hamilton-Jacobi Weighted ENO (HJ WENO)* FD scheme [51, 52, 69] takes a convex combination of the three possible third order accurate HJ ENO approximations at a given point. These weights can be chosen to obtain fifth order accuracy in smooth regions. In non-smooth regions they are set digitally (*i.e.* set to 0 or 1) so that we essentially revert to a HJ ENO, which attempts to avoid interpolating across discontinuities in the derivative. See the rightmost part of figure 2.5 for the stencil involved.

Both HJ ENO and HJ WENO result in significantly more complicated expressions

and have a higher computational cost than the first order approximations. However, by using them, the numerical accuracy and visual quality of the level set simulation is often greatly improved. For further details on both schemes and exact descriptions of their implementation we refer to [98] as well as the original papers.

Temporal Derivatives

For the level set equations we usually employ a *forward Euler* time step when first order temporal accuracy is sufficient. A forward Euler is essentially just a one-sided forward FD approximation in time: $\frac{\phi^{n+1}-\phi^n}{\Delta t}$. If higher temporal accuracy is needed, we turn to the *Total Variation Diminishing Runge-Kutta (TVD RK)* [112] schemes, which help diminish oscillations. As an example, consider the following third order accurate TVD RK scheme: First the current solution, ϕ^n, is advanced two steps forward in time using forward Euler to obtain ϕ^{n+2}. A weighted average is taken to form an approximation of $\phi^{n+\frac{1}{2}}$:

$$\phi^{n+\frac{1}{2}} = \frac{3}{4}\phi^n + \frac{1}{4}\phi^{n+2} \tag{2.8}$$

Then another forward Euler step is taken to obtain $\phi^{n+\frac{3}{2}}$, followed by a final weighted average:

$$\phi^{n+1} = \frac{1}{3}\phi^n + \frac{2}{3}\phi^{n+\frac{3}{2}} \tag{2.9}$$

Second or higher order accurate TVD RK schemes proceed in a similar manner. Generally, attempting to improve the temporal accuracy beyond the third order does not result in significantly better results since a spatial scheme such as HJ WENO reduces to third order accuracy in many of the interesting areas of the surface deformation.

2.3.2 Numerical Stability

In order to produce useful results, any numerical scheme for solving a PDE must be *convergent*. That is, the approximate solution computed by the scheme must converge to the exact solution as $\Delta x, \Delta t \to 0$. Convergence is in general hard to prove. Fortunately, the Lax-Richtmyer theorem states that convergence is equivalent to *stability* and *consistency*, and these properties are easier to prove [99]. Stability entails that the norm of the numerical solution at any point in time must be bounded by the sum of the norm of the numerical solution at a fixed number of earlier time steps. Therefore, without the stability requirement, the numerical solution of a given scheme could potentially blow up and grow uncontrollably. In the level set community, *explicit* numerical integration schemes are typically used to solve to level set equations. In such a scheme the approximation to ϕ^{n+1}, *i.e.* a grid point at time $t+\Delta t$, depends only on values at grid points from time t and/or earlier. This class of schemes usually has a *limited stability region*, which means that there is a restriction on the size of Δt given Δx, if a given method is to remain stable. In the following section, we will state the required restrictions on the time step, Δt, for each type of level set equation.

2.3.3 Solving the Level Set Equations Numerically

We will now focus on the numerical solution schemes for the level set equations stated in section 2.2. For clarity we will present the schemes in one dimension. The extension to higher dimensions can be performed in a simple component-wise manner. As we will see, despite the fact that equations 2.3 and 2.4 are mathematically equivalent, they require different numerical schemes. This is because equation 2.3 is linear in the partial derivates, while equation 2.4 is non-linear due to the term $|\nabla\phi|$.

Hyperbolic Level Set Equations

If the speed function and velocity field do not involve derivatives of the second order or higher, equations 2.3 and 2.4 are instances of *Hamilton-Jacobi* equations, a special class of *hyperbolic* PDEs [98]. Their solutions are constant along curves known as *characteristics*. The characteristic curves compose the *domain of dependence*, *i.e.* the part of the domain on which the exact solution in a given point depends. This property suggests a solution method to equation 2.3 known as *upwinding*. The *upwind* direction is the direction opposite to the direction of movement of a point on the surface. Intuitively, it is the direction from which information is flowing or emanating. Upwinding includes more grid points in the upwind direction than in the *downwind* direction when computing FD approximations to the spatial derivatives. For first order FD approximations this corresponds to choosing between one-sided forward or backward differences.

Assuming that we use a forward Euler time step for the temporal derivative, we can write up a numerical solution method for equation 2.3 based on the above observations:

$$\phi_i^{n+1} = \phi_i^n + \Delta t \left(\max(V,0)\phi_x^- + \min(V,0)\phi_x^+\right) \tag{2.10}$$

Note how the upwind scheme computes the derivative: If $V(x) < 0$ we use a forward difference, ϕ_x^+, and if $V(x) > 0$ we use a backward difference, ϕ_x^-. In both cases we are using upwind approximations of the derivative since we are favoring grid points from which the information is flowing. The actual computation of the derivative could be performed using a first order one-sided FD scheme or the higher order HJ ENO or HJ WENO schemes.

The above method is explicit, and it does indeed have a limited stability region as mentioned in the previous section. A necessary (but not sufficient) requirement for numerical stability is the *Courant-Friedrichs-Lewy (CFL)* condition [98]. For the above equation, it states that the time step must be restricted to $\Delta t < \frac{\Delta x}{\max\{|V|\}}$. In other words, the interface is restricted to move at most one grid point per time step since $\max\{|V|\}\Delta t < \Delta x$.

As previously mentioned, equation 2.4 is nonlinear because of the $|\nabla\phi|$ term. A widely used numerical scheme for computing this term is Godunov's scheme:

$$\phi_x^2 = \begin{cases} \max\left(\max(\phi_x^-,0)^2, \min(\phi_x^+,0)^2\right) & \text{if } V_N > 0 \\ \max\left(\min(\phi_x^-,0)^2, \max(\phi_x^+,0)^2\right) & \text{if } V_N < 0 \end{cases} \tag{2.11}$$

For three dimensions, ϕ_y^2 and ϕ_z^2 are computed analogously and the norm of the gradient is computed as $|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}$. Combining this approach with for instance a forward Euler time step, we obtain a method for solving equation 2.4. The CFL condition for this equation is more complicated, but it can be conservatively estimated by $\Delta t < \frac{\Delta x}{\max\{|V_N|\}d}$, where d is the dimensionality of the embedding function [98].

Noticing the similarities between level set equation 2.4 and the reinitialization equation (2.5), we see that Godunov's scheme can be readily applied to the reinitialization equation also. As mentioned, the reinitialization equation must be solved to steady state. In practice, we typically use a fixed number of iterations rather than checking for convergence [101]. Keeping in mind that the distance information is propagated in the normal direction with approximately unit speed, we can choose the number of iterations based on how wide a *band* of grid points around the zero isocontour we want to reinitialize. In the next chapter we will look at the narrow band level set method which only solves the level set equations in a narrow band around the zero isocontour.

For the sake of clarity, we used a first order forward Euler discretization of the temporal derivative in the above discussions. In practice, it is often necessary to use higher order methods such as second or third order TVD RK schemes.

Parabolic Level Set Equations

If the speed function or velocity field involves higher order partial derivatives, *e.g.* the mean curvature of the surface, the level set equations become *parabolic*. The parabolic equations have different mathematical, physical and numerical properties than the hyperbolic equations and therefore we must solve them differently. The solution at a given point in space does not flow along characteristics. Instead information flows into this point from all other points in space, and physically speaking, information travels at infinite speeds throughout the domain. This means that a perturbation in any part of space immediately influences all other points. In other words, the parabolic equation has an *infinite* domain of dependence. As an example, consider equation 2.4 with $V_N = -b\kappa$, where κ is the mean curvature and $b > 0$ is a positive scalar. This is a parabolic equation and for $b = 1$ it describes a smoothing operation on the surface. The parabolic equations have a stricter CFL condition which limits the time step to $\Delta t < \frac{\Delta x^2}{2bd}$, where d is the dimension [98].

The velocity field $\vec{V}$ in equation 2.3 is typically generated through some *external* computation, *e.g.* a water simulation, and therefore it does not depend directly on any differential properties of the surface itself. Hence, equation 2.3 is usually a hyperbolic equation, which we have already treated, and we will not consider it further here.

The Eikonal Equation

A widespread numerical method for solving the Eikonal equation (2.7), which is an *elliptic* PDE, is the first or second order accurate *Fast Marching Method (FMM)* [109,

127, 128]. The method is worst-case optimal with time complexity $O(N \log N)$, where N is the number of grid points in the grid. The algorithm starts by tagging the grid points on the zero isocontour, *i.e.* points which have at least one neighbor of differing sign, with a label called *Alive*. It then tags all points adjacent to the *Alive* set with a *Close* label. As it does so, it computes tentative distance values for these *Close* points using only *Alive* values and inserts the points in a heap sorted by these tentative values in an ascending order. The rest of the grid points are tagged *Far*. Now, the following steps are repeated iteratively until the *Close* set is empty: The top of the heap, *i.e.* the point with the smallest tentative distance, is removed from the heap and changes tag to *Alive*. Its final value is set to its current tentative distance, and all 1-neighbors have their tentative distances updated. Any *Far*-tagged 1-neighbors are changed to *Close* and are inserted into the heap with their respective tentative distances. When the algorithm terminates all points have been tagged *Alive* and therefore the distance at all grid points has been computed.

The central assumption for FMM is that the solution to the Eikonal equation (2.7) at each point only depends on points closer to the surface following the upwind characteristics. The ordering employed by FMM ensures that only these grid points contribute to the computation of the distance at a given point.

Recently, an alternative method called the *Fast Iterative Method (FIM)* [49] has been proposed. While related to FMM, the main idea of FIM is to avoid keeping track of the exact causal relationships as FMM does, and rather update the grid points in a looser manner explained below. This means there is no need for the heapsort of FMM. Instead FIM maintains a list of points which are currently being updated. This *active list* iteratively thickens and expands to include all points that could be affected by current updates. Points are then removed from it when they have converged with respect to their neighbors' current values. However, in constrast to FMM, points are put back into the active list when any upwind neighbor's value is updated. In detail, the algorithm proceeds as follows: First, the grid points on the zero isocontour are identified and their current values are locked as boundary conditions. All other grid points have their values set to infinity. Then, all 1-neighbors of the boundary are added to the initial active list L. After this initialization, the following steps are performed until L is empty: Each point has its value updated by solving the quadratic equation resulting from a Godunov upwind discretization (see [49] for details). If the solution has converged, *i.e.* the new value is sufficiently close to the old one, the point is removed from L and all non-converged 1-neighbors are added to the list. Note that newly added points must not be updated until the subsequent iteration.

FIM has a worst-case suboptimal time complexity because points may need to be added back into the active list if the characteristics change direction relative to the active list's propagation direction. In practice, this rarely happens when solving the Eikonal equation (2.7) with the surface as the boundary condition because the active list will propagate along the characteristics the majority of the time for most surfaces.

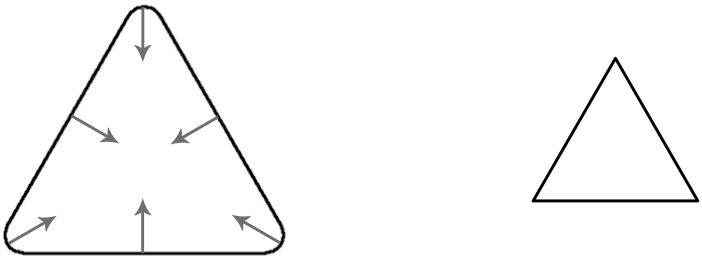


Figure 2.6 Left: Initial C^1 triangle with rounded corners propagating inwards with unit normal speed. **Right:** Non-differentiable triangle with sharp corners resulting from the propagation.

2.3.4 The Vanishing Viscosity Solution

Although the level set equations presented in the previous section are quite simple, and the accompanying numerical schemes introduced above are fairly easy to understand and implement, there are a lot of nontrivial issues in the underlying level set theory. One of the main problems encountered is the fact that, during interface deformations the physically reasonable (*i.e.* non-self-intersecting) solution is not always differentiable, even if the initial interface is smooth or just differentiable. Consider for example the left curve of figure 2.6. The distance function is C^1, but when it is propagated as indicated in the figure, the interface will at some point form sharp corners, where it is not differentiable as shown to the right in the figure. These three non-differentiable corners will continue to exist until the curve collapses to a single point. To solve these situations, it must be determined how to define the physically plausible solution mathematically and how to devise numerical methods that pick this solution automatically.

The desired solution can be defined by means of a so-called *entropy condition*. The solution obtained by adhering to this condition is a *weak solution*, which means that it is not necessarily an exact solution to the equation being solved, but rather a solution to an integral formulation of the equation, that agrees with the classical solution in all differentiable regions [98]. Since weak solutions are not unique, the entropy condition is needed to single out a solution. The solution picked is known as the *vanishing viscosity solution*, and it is obtained by adding a stabilizing, diffusive term $-\varepsilon\kappa$ to the right-hand side of the level set equations, where ε is the *artificial viscosity*. The inspiration comes from hyperbolic conservation laws where the method is used to avoid discontinuities (shocks) by smoothing the solution. It can be shown that this solution is identical to the desired weak solution in the limit, as $\varepsilon \to 0$. However, adding the artificial viscosity term explicitly when solving the level set equations numerically results in excessive smoothing of the solution. In fact, the numerical schemes themselves implicitly cause a smoothing effect known as *numerical dissipation*. Numerical dissipation can be understood by looking at the discretized schemes from a different viewpoint. Instead of thinking of them as producing an approximate, truncated solution to the level set equations, we can view them as producing an exact solution to a *slightly different* equation [99]. This equation, when derived, actually contains

a smoothing term where Δx acts as artificial viscosity. This explains why numerical schemes in practice introduce smoothing even though no artificial viscosity is explicitly added to the equations. It also tells us that the numerical dissipation tends to zero as the resolution of the grid is increased and $\Delta x \to 0$.

The numerical methods mentioned earlier have been devised so that they automatically pick out the physically plausible vanishing viscosity solution.

CHAPTER 3

LEVEL SET METHOD EXTENSIONS

We will now briefly touch upon some of the most important improvements that have been made to the original level set method [95]. Our focus will be on work that is relevant to computer graphics and the work presented in this dissertation, in particular. However, before we continue, we will outline the advantages and disadvantages of the level set method in order to motivate both its use and the extensions presented in the following. So far no "silver bullet" surface representation has been proposed, but rather all surface representations have their own unique advantages. Nevertheless, as explained in the previous chapter, level sets are very powerful and well-suited for physically based animation. Some of the main advantages of level sets are that they:

- avoid self-intersecting geometry.
- handle arbitrary topological changes.
- avoid aliasing and singularities when undergoing large deformations.
- allow for easy computation of differential properties such as the gradient, surface normal and curvature to a high order of accuracy.
- trivialize geometric queries such as inside/outside tests and closest point transformations.
- allow for easy computation of *Constructive Solid Geometry (CSG)* operations such as union, intersection and difference of solids.
- can be rendered directly using ray tracing.

There are a number of significant disadvantages to the original level set method. While a number of them have been largely resolved by recent and ongoing work — as we will see in the following section — we still list them here for completeness:

- The computational requirements scale with the volume of the embedding function.

- The storage requirements scale with the volume of the embedding function.
- Even though the first bullet point above has been addressed by restricting the solution of the level set equations to a narrow band around the surface, accurate level set methods (*e.g.* HJ WENO with TVD RK) are still very time-consuming compared to mesh based deformations. This is due to the fact that the relatively expensive PDE computations must be performed at each grid point[1].
- In order to maintain sharp edges and features, high resolutions combined with high order accurate FD schemes are usually required. In some situations, this approach can be replaced or combined with Lagrangian marker particles (see section 3.4), but generally level sets are computationally intensive and simulations do not typically run at interactive rates except at small resolutions or when running on the GPU [64].
- Many passes over the surface are performed in order to move the surface and reinitialize the embedding function. This is a problem when the level set is too big to fit in main memory since it must be streamed to and from external memory (*e.g.* the hard disk) many times in just one time step.
- Since triangle meshes are still the most widely used surface representation, it is often necessary to convert back and forth between level sets and meshes. Unfortunately, current methods for doing that are not invertible in the mathematical sense.
- Explicit surface representations such as meshes and subdivision surfaces can be adaptive along the surface, whereas level sets are still represented uniformly.
- In contrast to explicit surface representations, level sets have no inherent parameterization. Parameterizations are useful for many applications in computer graphics, *e.g.* texturing and other kinds of two-dimensional mappings.

3.1 Narrow Band Level Set Methods

In the original level set formulation the computational complexity is $O(N^3)$ where N is the number of grid points in each dimension of the three-dimensional grid. Often one is only interested in the two-dimensional surface embedded in the grid and therefore this approach seems highly inefficient. One of the first major improvements was to exploit the property that only grid points close to the zero isocontour are required to compute the movement of the surface. Therefore one can confine the computations to a *narrow band* around the surface. The idea has been refined by a number of authors [1, 16, 93, 101, 132]. The method improves the computational complexity to be $O(S\delta)$ where S is the area of the surface while δ is the width of the narrow band in grid points. It is important to note that the narrow band should be wide enough to support the

[1] Not all applications of level sets require higher order accurate FD schemes to give visually pleasing results, however. Shape morphing is one example which can often be sufficiently simulated using first order upwind schemes.

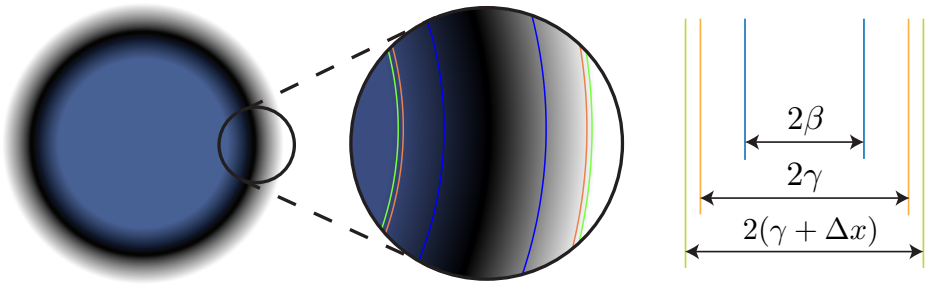


Figure 3.1 **Left:** Narrow band level set with values outside the narrow band clamped to $\pm\gamma$. **Middle:** Close-up of the concentric tubes which make up the narrow band. The β tube is shown in blue, the γ tube in orange and the entire narrow band in green. **Right:** The value ranges covered by each of the tubes.

stencils of the FD schemes employed for the spatial derivatives at the zero isocontour. Also, the necessary width depends on the number of grid points the surface can move during one time step. Recall from the previous chapter, that most explicit methods for stability reasons are restricted in terms of the time step size they can employ. The CFL condition implies that the surface can move less than one grid point in each time step.

In the following, we will concentrate on the narrow band method of Peng *et al.* [101] along with a more recent extension by Nilsson *et al.* [93] which improves the computational efficiency. We will assume that the surface moves at most one grid point (*i.e.* at most Δx) in each time step. The method of Peng *et al.* represents the narrow band as three concentric tubes around the zero isocontour. The middle of figure 3.1 shows the approach. The narrow band is equal to the largest tube of width $\gamma + \Delta x$ and inside it are two narrower tubes called the γ and β tubes, respectively. The values inside the γ tube are signed distances, while the values outside are clamped to $\pm\gamma$. This means that the unique properties of the signed distance function pointed out in the previous chapter are only available inside the narrow band. Outside the narrow band, only the clamped signed distance is available[2]. For first order upwind schemes $\beta = 2\Delta x$ and $\gamma = 3\Delta x$ whereas for a fifth order HJ WENO scheme $\beta = 3\Delta x$ and $\gamma = 6\Delta x$ [101].

In order to obtain a computational complexity linear in the number of grid points in the narrow band, a few extra data structures are utilized by the narrow band method. First of all, an array containing the coordinates of the grid points in the narrow band is used. This enables iteration over the grid points in the narrow band in linear time. Secondly, a grid of the same dimensions as the simulation grid is maintained. The values of this grid indicate for each grid point which tubes it lies within, if any.

The actual computations performed by the narrow band level set method are largely similar to the original method presented in section 2.3.3. In particular, the advection of the surface by solving the level set equations 2.3 or 2.4 proceeds as normal within the β tube. For grid points only inside the γ tube (*i.e.* points $\mathbf{x}$, where $\beta \leq |\phi(\mathbf{x})| < \gamma$), the solution is modified by a cut-off function to avoid numerical oscillations at the

[2] Narrow band level sets are however still useful for *e.g.* ray tracing and collision detection.

boundary of the narrow band. Peng *et al.* [101] suggest the following function $c(\phi)$:

$$c(\phi) = \begin{cases} 1 & \text{if } |\phi| \leq \beta \\ \frac{(|\phi|-\gamma)^2(2|\phi|+\gamma-3\beta)}{(\gamma-\beta)^3} & \text{if } \beta < |\phi| \leq \gamma \\ 0 & \text{if } |\phi| > \gamma \end{cases} \tag{3.1}$$

Note that $c(\phi)$ transitions smoothly from one to zero inside the $\gamma-\beta$ tube. For grid points outside the γ tube, the level set equation is not solved.

After advection, the reinitialization equation (2.5) is solved to steady state in the *entire* narrow band as described in section 2.3.3. Hence after reinitialization, the γ tube of the *advected* surface will be contained within the narrow band. This is due to the fact that the surface moves at most one grid point during a time step and that the narrow band was constructed to be exactly one grid point wider than the γ tube.

Next follows a computation which is exclusive to the narrow band method. It is necessary to rebuild the narrow band and ensure it follows the surface correctly. In other words, we need to compute the β, γ and $\gamma+\Delta x$ tubes again. The original proposal by Peng *et al.* [101] used a simple method with a computational complexity of $O(N^3)$. They argued that the rebuild process is performed only once per time step whereas advection and reinitialization entail several iterations over the narrow band. Even so, the $O(N^3)$ time complexity can become dominant when the grid is large, and a method which scales with the number of grid points in the narrow band is preferable. Nilsson *et al.* [93] proposed the following method which determines the new narrow band using a single pass over the old narrow band: For each grid point $\mathbf{x}$ in the old narrow band, if $|\phi(\mathbf{x})| < \gamma$, $\mathbf{x}$ is included in the new narrow band. If $\mathbf{x}$ was not included in the old γ tube, all neighbors of $\mathbf{x}$ not already in the new narrow band are added to it. On the other hand, if $|\phi(\mathbf{x})| \geq \gamma$, all neighbors $\mathbf{y}$ of $\mathbf{x}$ with $|\phi(\mathbf{y})| < \gamma$ are included in the narrow band.

To round this section off, we note that, while the narrow band methods presented above address the computational inefficiency of the original level set formulation, they *increase* the storage requirements of the original method since extra data structures are required to keep track of the narrow band. This is an issue for level set simulations that require high resolution grids to resolve complex deforming surfaces.

3.2 Octree Based Level Set Methods

Octrees (and *Quadtrees* in two dimensions) have been applied to level sets [72, 117] and adaptively sampled distance fields [35] in order to reduce the computational and storage requirements of the original level set method. These tree data structures allow for adaptive multi-resolution representations. However, all current tree based level set methods use a uniform resolution near the surface. This is mainly because it can be hard to design reliable refinement strategies which guarantee that fine details and features are not lost due to undersampling as the surface moves in time. Also, a uniform refinement near the interface, which only stores the grid points inside the narrow band in the octree, enables the use of higher order FD schemes like HJ ENO [96] and HJ

WENO [51, 52, 69] in space. In contrast, a nonuniform discretization makes it non-trivial to accurately employ these FD schemes. Tree based methods which solve the level set equation over the entire embedding domain need to account for grid cells of varying sizes and are hence not able to use these FD schemes as is [77, 117].

The octree data structure reduces the storage requirements of level sets to $O(dS)$, where d is the depth of the tree and S is the surface area. However, it also introduces an $O(d)$ access time to the stored values. Note that it is quite possible that $d \gg \log S$. The data structure can be modified to reduce storage requirements to the asymptotically optimal $O(S)$, and access times to $O(\log S)$ [117]. Nonetheless, a logarithmic access time is still costly in the context of the level set method. Furthermore, modern octree-traversal and search methods [34] employ bit-arithmetic that cannot readily be used with these modifications.

Losasso *et al.* [72] recently proposed a method which addresses some of the performance issues associated with octrees. Rather than using a regular octree they propose the use of a coarse uniform grid in which each grid cell stores its own octree. In this way they decouple the depth, d, of the octree from the size of the embedding domain, thereby lowering d.

Combining the narrow band method with a level set representation based on an octree thus potentially promises an asymptotically optimal method with regards to storage [117]. However, the underlying pointer structure employed by an octree often entails random access in memory which is not suited for the memory hierarchies of modern computers [11]. Also, even if the initial octree is constructed carefully to support coherent memory access patterns, no method has been published on how to efficiently ensure cache coherency in the octree storage format as the narrow band is updated over time due to the evolution of the surface. We will elaborate upon the issue of cache coherency and memory hierarchies in section 4.1. Finally, since each access to a value stored in the octree has a time complexity which is linear in the depth d, this approach is so far generally on par with or outperformed by the next class of methods.

3.3 Sparse Non-Tree Based Level Set Methods

Bridson suggested using a sparse, coarse, uniformly blocked grid with finer uniform grids nested inside those coarse cells that are close to the surface [11]. In particular, the entire embedding domain is split into blocks of B^3 grid points each, and a coarse grid of size $\left(\frac{N}{B}\right)^3$ stores pointers only to those blocks that are intersecting the narrow band. As the surface moves, blocks are allocated and released. Because of the fixed depth of the hierarchy, access times are $O(1)$ as for dense grids. The storage complexity is reduced to $O(\left(\frac{N}{B}\right)^3 + S)$ which allows for higher resolutions as long as the storage requirements of the coarse grid, $\left(\frac{N}{B}\right)^3$, are $O(S)$. In that case the storage complexity is $O(S)$ which is ideal. However, for very high resolutions or for simulations where surfaces with small areas are spread out in a large embedding domain, the storage requirements may be dominated by $\left(\frac{N}{B}\right)^3$. The method restricts computations to a blocked narrow band, which is a fairly conservative estimate of the narrow band actually needed. This means

the method will perform more computations than necessary if the grid points actually inside the narrow band are not explicitly marked using some extra data structure for each block. Hence, although this representation provides constant time access to values, it does not always ensure storage and computational requirements proportional to the area of the surface.

Nielsen and Museth [90] introduced the *Dynamic Tubular Grid (DT-Grid)* which has the advantage that both the computational complexity and storage requirements scale linearly with the size of the interface (as opposed to the volume of the embedding function). This data structure is the foundation of the first part of the work presented in this dissertation. It takes its inspiration from sparse matrix storage schemes, and exploits the fundamental property of narrow band level set methods that the level set equation only needs to be solved close to the surface. However, unlike previous narrow band methods, it does not store *any* information outside a dynamic, uniformly sampled narrow band. Note that the DT-Grid stores the values and the topology of the narrow band separately. The uniformity inside the narrow band makes it possible to readily employ all FD schemes developed for dense uniform grids. The DT-Grid is defined recursively by DT-Grids of lower dimensionality. Therefore it easily generalizes to any number of dimensions. Each level stores the *connected components* of the corresponding dimension rather than all the grid points. Please see the original paper for the definition of a connected component and a more thorough explanation [90]. The DT-Grid can furthermore be extended to accommodate volumetric attributes for fluid simulations in which case the scaling is proportional to the volume of the fluid. Also, it is not bound by the boundaries of an underlying grid and is free to dynamically expand as the level set grows. The data structure is designed for sequential access. Particularly, the grid points in the narrow band can be accessed in asymptotically constant time, $O(1)$, when using a sequential access pattern. On the other hand, random access is logarithmic in the number of connected components in each dimension, but fortunately the DT-Grid comes with a set of iterators designed to provide constant time access to all the grid points required by a given FD stencil. This means that solution methods to the level set equations, such as those presented in the previous chapter, can be applied in linear time. The same goes for solving the reinitialization equation, and an algorithm for rebuilding the narrow band in linear time is also provided in [90]. However, there is no support for random insertion or deletion of grid points in the narrow band. On top of the linear algorithms and constant sequential access times, it appears that the memory layout and footprint of the DT-Grid makes it very cache coherent.

Concurrently with the development of the DT-Grid, Houston *et al.* presented the *RLE Sparse Level Set* representation in a technical sketch [46]. The idea of using *Run-length encoding (RLE)*[3] for compressing level sets was first proposed by Bridson [11], but Houston *et al.* were the first to design and implement a concrete data structure. It is more flexible than the DT-Grid because it allows for an arbitrary encoding of the narrow band, including varying grid cell sizes, and their method is also able to store open level sets. The drawback is that it requires $O(N^2+S)$ storage in three dimensions.

[3] RLE is a widely used lossless data compression technique in which sequences (runs) of identical data values are replaced by a count number and a single value.

Also, the sequential access times are not constant per element. This was remedied by the *Hierarchical RLE (H-RLE)* grid [47] which retains the flexibility of the original RLE approach, but has the same asymptotical behavior as the DT-Grid. In practice, however, it is a little slower, but depending on the application the increased flexibility might be worth the small extra cost.

In parallel to this dissertation work, Museth has developed an efficient blocked level set data structure, dubbed *DB+Grid*. Given the fact that DB+Grid is still proprietary technology, our insight into this data structure is limited. However, for the sake of completeness, we include the following high-level description, and refer the interested reader to an upcoming publication. DB+Grid is based on a *Dynamic Blocked Grid* that exploits the *spatial coherency* of uniform grids to effectively and separately encode data values and topology. It shares several characteristics with B+Trees (typically employed in databases and file systems), which accounts for its name. DB+Grid allows for cache-coherent and fast data access into sparse three-dimensional grids of very high resolutions, *i.e.* exceeding hundreds of thousands of grid points in each spatial direction. Additionally, DB+Grid is very general since it does not impose topology restrictions on the sparsity of the volumetric data or access patterns when the data are inserted, retrieved or deleted. This is in contrast to both DT-Grid and H-RLE that assume fixed data topology (narrow band level sets), and require specific access patterns (sequential) for fast data access. As such, DB+Grid has proven useful for several applications that call for very large sparse volumes, and it has already featured in several major films [81, 82, 85].

3.4 Particle Level Set Methods

A way of achieving a high level of detail on a fairly coarse grid, is the *particle level set* method, which combines Lagrangian marker particles with the Eulerian level set method to obtain the advantages of both approaches [23]. The particles reduce the numerical dissipation which is inherent in the level set method, especially at coarse resolutions. They ensure greater accuracy in regions of high curvature, where the vanishing viscosity solution of the level set method would incorrectly merge the characteristics. Lagrangian schemes are therefore quite successful in conserving mass, since they preserve material characteristics for all time, *i.e.* they are not regularized out of existence to obtain vanishing viscosity solutions. On the other hand, the Eulerian level set method ensures that changes in the surface topology are correctly resolved. These properties have made the particle level set method the widely preferred surface tracking method for water simulations within computer graphics. The method is perfectly suited for situations where the surface is passively advected in an external shock-free velocity field. However, its use beyond this application area is limited. First of all, it does not work when shocks[4] are present in the velocity field since they will cause the particles to move inconsistently with the surface and hence ruin the desired weak solution [23]. For the same reason, the particle level set method is not well-suited to surface deformations described by motion in the normal direction (equation 2.4). This

[4] In the vicinity of shocks, characteristics actually *do* merge.

means that the method is impractical for many applications in computer graphics such as shape metamorphosis and geometric modeling that employ motion in the normal direction and have shocks in the underlying velocity field where the model has sharp corners or edges.

The particle level set method increases the memory requirements as up to 64 particles are placed in each grid cell in a band that is three cells wide on each side of the surface. For each particle, both position and radius are stored. As the surface area increases, the memory requirements follow, and even though a quantization of both position and radius somewhat alleviates the problem, a lot of memory is still needed. Therefore *out-of-core* or *external memory* algorithms[5] have been developed [91]. To sum up, there are currently no attractive alternatives to turning up the resolution when one wants to obtain a high level of detail for *general* level set simulations [11], and for surface deformation in fluid simulations, out-of-core algorithms are necessary to handle the enormous amount of particles required. Therefore out-of-core algorithms have become an important part of making level set simulations and fluid simulations feasible in practice.

3.5 Out-of-Core Level Set Methods

A common limitation of all the described methods is that they only work *in-core*, *i.e.* they are limited by the available main memory. As a consequence, implicit model resolutions allowed by the methods are quite small compared to state-of-the-art explicit representations, despite the improvements in storage requirements. This in turn means that the level of detail obtainable by these methods is not nearly as high. Surprisingly, given these limitations of the described methods, little work seems to have been done to design external memory algorithms for level set methods and fluid simulations. Only recently, Nielsen *et al.* [91] proposed an out-of-core framework for level set and fluid simulations. The main contributions are prefetching and page-replacement algorithms designed for stencil based FD level set computations. The framework is generic and can be combined with several different kinds of underlying level set representations such as the DT-Grid or H-RLE sparse data structures, which means that it can easily be integrated with existing software. In particular, it does not require a modification of the level set algorithms associated with these representations. While the framework allows for grid resolutions only limited by the available secondary storage, the method remains I/O bandwidth limited, and the throughput (measured in computed grid points per second) drops as low as 42% of state of the art peak in-core simulation throughput, when all components of a DT-Grid must be streamed to and from disk. This is primarily due to the fact that the level set is streamed several times during a single simulation time step. Although the throughput is an improvement over general purpose OS prefetching and page-replacement algorithms, the framework presented in chapter 4 and paper I demonstrates a throughput as high as 92%.

[5] Out-of-core algorithms are designed to process data that is too large to fit in main memory (*in-core*) at once. They need to be carefully designed in order to efficiently fetch and access data stored in slow bulk memory such as a hard disk.

CHAPTER 4

THE IMPROVED OUT-OF-CORE FRAMEWORK

Chapter 2 introduced the level set method and motivated its use in computer graphics. In particular, we emphasized the importance of level sets for surface representations in fluid simulation, shape metamorphosis and geometric modeling. A number of issues with the original formulation, which limit the practical applicability of level sets, were identified in chapter 3, and we presented several extensions designed to address them. However, as mentioned at the end of the previous chapter, there are still some problems left unsolved. Our work presented in this chapter and paper I targets two of the problems which are mainly related to level sets of high resolution[1]. First of all, the typical approach to performing a time step in a level set simulation requires multiple passes over the surface. Especially solving the reinitialization equation (2.5) may involve up to 5 – 10 passes over the entire level set. However, simply employing higher order temporal schemes for propagation such as TVD RK also involve several passes. This becomes a problem since the level set data must be streamed through memory several times, which could impact on cache locality. When the level set is too large to fit in memory, the problem becomes even worse and it is actually one of the main reasons for the I/O limitation in [91] since the level set is streamed to and from disk multiple times in a single time step. In contrast, the method proposed in paper I requires data to be streamed only once per time step. In fact, for simulations with certain properties, data is only required to be streamed once for a number of subsequent time steps, hence reducing I/O bandwidth usage further. We do this by applying *code transformations* such as loop *skewing* and *tiling* to the level set computations, see section 4.2. As a result, our new out-of-core algorithms are CPU limited as opposed to I/O limited and exhibit a sustained, *i.e.* resolution independent, throughput of 77% – 92% (depending on the numerical scheme) of the throughput obtainable by internal memory simulations.

Secondly, we target the fact that high-resolution level sets are computationally heavy. We do this by enabling individual *tiles* of the level set to be processed independently, hence allowing for multi-threading. Because of the reduction of I/O bandwidth

[1] In order to provide a coherent reading experience some of the body of paper I will be repeated in this chapter.

usage resulting from our transformed computations, the method is still CPU limited when run on 8 cores, even though each thread writes to and reads from disk. Of course, this means that running multiple simulations in parallel on the same disk is feasible too.

Our improved out-of-core framework is essentially based on a couple of techniques that are well known in the field of computer science, and in the fields of compiler algorithms and cache coherency in particular: We mainly rely on code transformation methods such as loop skewing, tiling, interchange and fusion. As such there is a large body of related work which is presented in paper I. However, we stress that our work stands apart from this previous work in several ways. Most importantly, we are the first to optimize and apply these techniques on out-of-core level set methods. Of course, the original framework leveraged over fifty years of active research into out-of-core methods. However, since we have merely built on top of that part of the framework, we refer the interested reader to the original paper [91] and the recent surveys on out-of-core methods by Vitter [130] and Toledo [125] for a more detailed overview of that field and its contributions to numerical linear algebra and simulation.

The rest of this chapter is structured as follows. Section 4.1 briefly introduces the typical memory hierarchy of a modern computer as well as a couple of models for analyzing algorithm performance that take this hierarchy into account. Next, section 4.2 describes the concept of *code transformations* which essentially reorganize program loops in order to ensure a better cache locality. The subsequent section summarizes the contributions of paper I and the last section discusses and evaluates the results.

4.1 Memory Hierarchies and Cache Locality

Generally speaking, faster memory is more expensive than slower memory. Additionally, larger memory entails a higher latency due to the increased number of transistors. To provide the best performance at a reasonable cost, memory in modern computers is organized in a hierarchical fashion, where each level acts as a cache for the next. The typical components of this hierarchy are registers, L1 through L3 caches, main memory and disk storage. Figure 4.1 shows a simplified view of this hierarchy along with some rough numbers illustrating how storage capacity, access speed and price vary throughout the levels. To access a particular piece of data, the CPU first sends a request to its nearest memory, usually the caches. If the data is not in any of the caches, then main memory is queried. If the data is not in main memory, then the request goes to disk. Once the data is located, it and a number of its nearby data elements are fetched into cache memory. The reasons for loading these elements into cache are based on two observations about normal program behavior. First of all, many programs exhibit *spatial locality*, which means that memory locations close to a recently accessed location are likely to be accessed in the near future. Secondly, many programs carry *temporal locality*, where a recently accessed memory location is likely to be accessed again. Oftentimes, this is seen in numerical simulations where the same data elements are repeatedly treated for several iterations or time steps. The L1 through L3 caches store fixed-size blocks of the main memory in *cache lines* which consist of a number

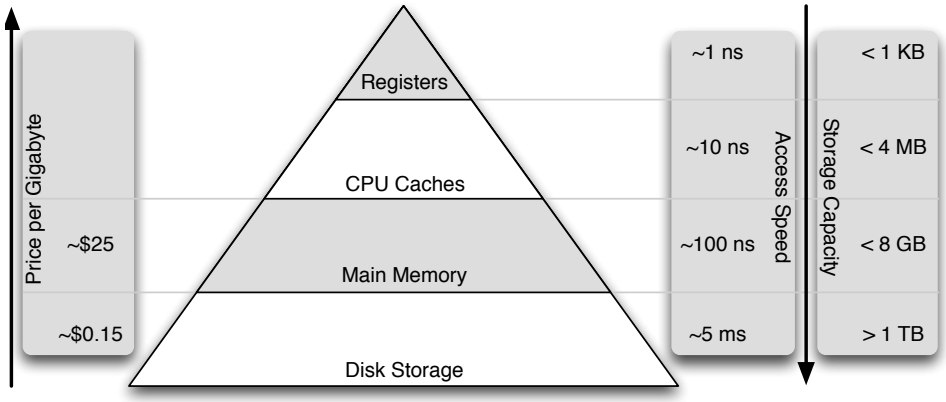


Figure 4.1 A simplified view of the typical memory hierarchy of a desktop computer. The numbers are very rough approximations and are changing rapidly, but they give an idea of the ratio between the levels.

of *words*. If a given word is not in cache, an entire cache line is evicted from it and replaced by the cache line which holds the word. The question of which cache line to evict is very important, since the time for accessing a level increases for each level, which means that the speed with which the data can be accessed depends on which level currently holds it. When the data is found at a certain level, we say that we have a *hit*. On the other hand, a *miss* occurs when the data must be found at a lower level. The fraction of all memory references that can be satisfied by a given level is called the *hit ratio* of that level. Larger caches have better hit ratios but longer latency.

Similar to the relationship between words and cache lines of the main memory, data elements are transferred from disk to main memory as *pages* which are read from disk into the virtual address space of a running process. If a given piece of data is not found in any of the pages, a page fault is generated and the page is loaded in from disk. Of course, this often means that another page must be evicted and several page replacement schemes exist. We use the scheme from [91], which is specialized for level set computations. Note that the latency for accessing external memory is four to six orders of magnitude slower than accessing main memory. Also note, that the amount of data transferred when accessing external memory is many orders of magnitude larger than when loading a cache line. This means that optimizing for *cache locality* between main and external memory is quite different from targeting CPU caches and main memory. Further information on memory hierarchies can be found in the excellent book by Tanenbaum [120].

As is evident from the above discussion, the memory access pattern of an algorithm has a vital influence on its efficiency. Asymptotic analyses in the traditional RAM model [38] are unable to capture this aspect, and so a number of more involved models have been proposed. The most influential of these are the *I/O model* introduced by Aggarwal and Vitter [2] and the more recent *Cache Oblivious model* by Frigo *et al.* [33]. The I/O model simplifies the memory hierarchy to two levels consisting of the main memory and one disk. It is parameterized by the number of data elements that fit into memory, M, and the number of elements in a single disk block, B. It can be used

to explore the fundamental limits in terms of the number of I/O operations required for various typical computational problems such as sorting. Linearly scanning through N data elements hence requires $\Theta(\frac{N}{B})$ I/O operations. If B is small, the I/O complexity will dominate the running time of the algorithm. Sorting N elements has been proven to have an I/O complexity of $\Theta(\frac{N}{B}\log_{1+\frac{M}{B}}(1+\frac{N}{B}))$ [2]. Having these values for a given architecture one can attempt to design an algorithm which results in the best I/O complexity. However, that algorithm may not run as efficiently on another system with other values for M and B. Also, the model completely ignores the other levels of cache in the memory hierarchy.

The Cache Oblivious approach [33] attempts to overcome this limitation by abstracting away the concrete parameters of a given system configuration. By doing so it ensures that algorithms which perform well in this model are efficient across different architectures and on all levels of the memory hierarchy. Specifically, it introduces the *ideal-cache model* which is comprised of a two-level memory hierarchy consisting of an ideal cache of size Z and an arbitrarily large main memory. The model assumes an optimal replacement strategy of the cache lines[2], which have size L. Algorithms are analyzed in two ways in the model: Their work complexity $W(N)$ is similar to the traditional RAM running time complexity. In addition, they have a cache complexity $Q(N,Z,L)$ which is a measure of the number of cache misses the algorithm will experience. A good *cache oblivious algorithm* attempts to match the work complexity of some RAM model algorithm while minimizing Q. Furthermore, it should be independent of the cache specific parameters Z and L, hence the term cache oblivious. As stated, a cache oblivious algorithm will likely work efficiently across different architectures and on all levels of the memory hierarchy. However, the simplifying assumptions may hide important properties, and we cannot expect a cache oblivious algorithm to be as efficient in practice as a *cache aware* algorithm specifically tailored for a given architecture.

4.2 Code Transformations

The algorithms presented in paper I leverage on established theory from the area of compiler algorithms. They rely on code transformations for optimizing cache locality, *i.e.* minimizing the number of times a given data element is loaded from the lowest memory level into the highest level of cache. In particular, we employ the mathematical model of reuse and locality developed by Wolf and Lam [134]. In the class of code transformations necessary to consider for FD based level set schemes, the execution of a particular program statement in an algorithm can be identified uniquely by a node inside a convex polytope embedded in an iteration space. Considering a simple for-loop with iteration variable i surrounding a single statement, the iteration space equals $\mathbb{Z}$ and the polytope consists of the nodes in $\mathbb{Z}$ delimited by the lower and upper limits of i. Nesting one for-loop within another adds one to the dimensionality of the iteration space and in general a loop nest of depth n corresponds to

[2] In practice, such a strategy is not possible without knowing the entire future of a program execution. However, using the *Least Recently Used* strategy [120] may give results which are close to optimal depending on the program.

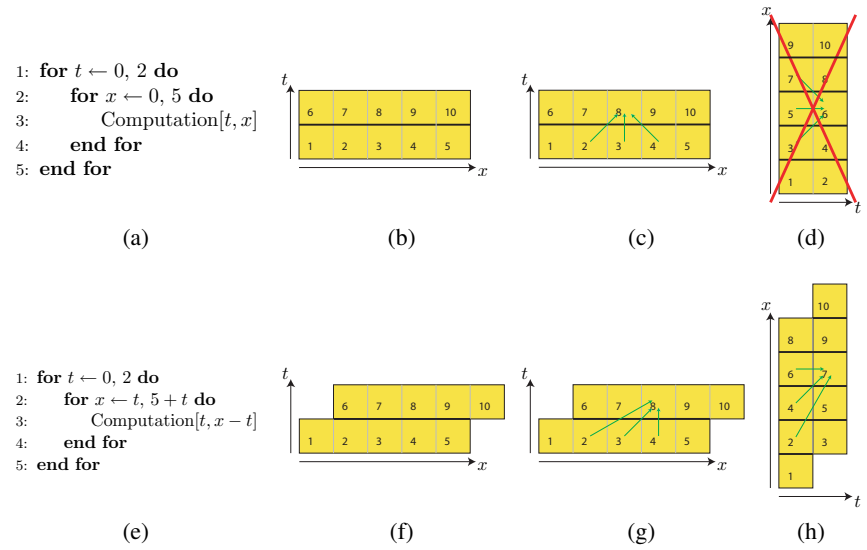


Figure 4.2 **(a):** Pseudo-code for a simple loop nest with iteration variables t and x. **(b):** A convex polygon covering the computation nodes in the two-dimensional iteration space spanned by the loops in t and x. **(c):** The green arrows indicate that a given computation $[t,x]$ depends on the computations $[t-1,x-1]$, $[t-1,x]$ and $[t-1,x+1]$ which therefore must be performed first. **(d):** As indicated we are not able to interchange the loops. **Bottom:** The loop has been skewed by the transformation $T : [t,x] \rightarrow [t,x+t]$. Note how the loop interchange in (h) no longer violates the dependencies.

a convex polytope embedded in $\mathbb{Z}^n$. Executing a loop nest thus corresponds to visiting all statements or — equivalently — all nodes in the polytope in lexicographic order[3]. A valid code transformation T, *e.g.* a linear transformation, transforms a polytope without violating data dependencies. That is, if node p in the polytope depends on data produced at node q, then $T(q) < T(p)$ must hold, where $<$ is the lexicographic ordering. Figure 4.2 shows a simple two-dimensional example with iteration variables t and x. The corresponding polygon in the iteration space is shown in 4.2b. The data dependencies of Computation$[1,2]$ are shown as green arrows in 4.2c. They indicate that Computation$[t,x]$ reads the values from Computation$[t-1,x-1]$, Computation$[t-1,x]$ and Computation$[t-1,x+1]$. For the purpose of our example, suppose we have a cache that can only hold four data elements and that a cache line consists of one data element. The value from Computation$[0,0]$ referred to by for instance Computation$[1,0]$ is therefore evicted between references. Unfortunately, a loop interchange represented by the code transformation $T_1 : [t,x] \rightarrow [x,t]$ is not possible as illustrated in 4.2d. The reason is that the data dependencies of Computation$[1,x]$ are violated as they depend on Computation$[0,x+1]$ which is performed afterwards. However, if we apply the transformation $T_2 : [t,x] \rightarrow [t,x+t]$ which *skews* the polygon as illustrated in 4.2f, we also change the dependencies. In fact, this skewing allows us to interchange the loops without violating the dependencies of the algorithm as illus-

[3]Lexicographic order is akin to alphabetical order, but applied to the Cartesian product of ordered sets. It is defined for two sets as $(a,b) < (a',b')$ iff $a < a'$ or $(a = a'$ and $b < b')$, and it generalizes straightforwardly.

trated in figure 4.2h. Note, that our cache can now hold the values needed at any given time in order to perform the computations and therefore each data element is only fetched into the cache once. Thus skewing does not necessarily improve the performance of the code by itself but it allows for other code transformations to be performed subsequently.

Wolf and Lam [134] quantify the *reuse* of a data element by the number of directions in the iteration space along which the algorithm references that particular data element or data elements that are close when stored in memory (*i.e.* on the same cache line). A high degree of reuse does not however imply a high degree of cache locality or high performance. The reason is that if access is not cache coherent (*i.e.* too many other data elements are accessed in between), in the worst case a data element may have to be loaded into the cache each time it is used. In order to ensure locality, we must divide the iteration space into *tiles*. This *tiling* transformation is performed by dividing the iteration along each direction of the iteration space by splitting the corresponding loop into two loops, where the loop that steps over the tiles is called the controlling loop. As we will see in appendix A of paper I, it is *necessary* to skew level set computations in order to make them fit for tiling.

Despite the fact that code transformations have been extensively studied and employed for optimizing cache locality in compiler algorithms, the work presented in paper I is to the best of our knowledge the first to apply and study code transformations in the context of out-of-core level set simulations. Applying code transformations to out-of-core level set simulations poses unique challenges since the goal here is to minimize the data traffic, *i.e.* maximize locality, between disk and main memory, as opposed to between main memory and the CPU caches. The ratio of disk to main memory latency is high relative to the ratio of main memory to cache latency. Consequently, a data layout working well in-core may have to be designed differently for out-of-core use (although it may have the same asymptotical I/O complexity) in order to efficiently exploit the often large page sizes utilized when transferring data between disk and main memory.

4.3 Contributions

The primary design goal of our out-of-core level set framework has been to perform an entire time step of a level set simulation using only one strictly sequential pass over the level set data. Recall that one of the main reasons for the I/O bandwidth limitations of the previous out-of-core framework proposed by Nielsen *et al.* [91] was that each time step required up to 10 passes over the data. The out-of-core framework proposed in paper I has several benefits, both theoretically and practically. Specifically our framework is optimal with respect to streaming to and from disk in both the I/O model [2] and the Cache Oblivious model [33], as only sequential stencil access is required. Since the DT-Grid is utilized as the underlying data structure, our framework inherits its properties such as memory efficiency, low L1 and L2 cache miss ratios, constant time access to grid points in an FD stencil, generalization to any number of dimensions as well as the benefit of a dynamically expanding and contracting grid not

confined to a predefined computational domain. Similar to the out-of-core framework proposed by Nielsen *et al.* [91] the resolution of the computational grids utilized in our method is only limited by the amount of secondary storage, *e.g.* disk space, available.

Note that while the description of the framework in paper I exclusively focuses on narrow band solutions of interface propagations represented as high-resolution level sets, the methods presented generalize to arbitrary, sequential stencil computations on full grids as well. Specifically, it should be noted that the presented framework can be used for fluid simulations, although only in a somewhat limited way as will be explained in section 3.1.5 of the paper.

To summarize, the work presented in paper I claims the following contributions:

- A combination of code transformations and storage mapping schemes with the DT-Grid [90] and the prefetching and page-replacement schemes for out-of-core computation proposed in [91]. This allows data to be streamed to and from memory only once during each iteration of the out-of-core level set computation, where each iteration comprises a propagation/advection, a reinitialization and a narrow band rebuild step. For some types of simulations, the transformations of N subsequent iterations can be concatenated, hence requiring data to be streamed to and from memory only once for each group of N iterations.
- A tiled version of the *Fast Iterative Method* [49] which enables a fast, out-of-core solution of the Eikonal equation (2.7) for narrow band level sets.
- Code transformations, based on skewing and tiling, of FD based numerical level set schemes, including Forward Euler, TVD RK, HJ ENO and HJ WENO. We prove that, both with respect to the spatial and temporal dimensions, the transformations maximize the locality of references in the mathematical model of reuse and locality proposed by Wolf and Lam [134]. Our framework is optimal with respect to streaming to and from disk in both the I/O model [2] and the Cache Oblivious model [33] as only sequential stencil access is required.
- In-core storage mapping schemes for intermediate values associated with the skewing transformations that reduce the memory requirements during computation. Specifically, the storage complexity is linear in the number of intermediate values.
- An out-of-core storage mapping scheme associated with the tiling transformations that partitions the narrow band into individual grid blocks and additional boundary grids. This facilitates computation on each tile independently and hence in parallel. The storage complexity remains $O(N)$ when combined with the DT-Grid, where N is the number of grid points in the narrow band.

Read paper I at this point.

4.4 Further Discussion and Evaluation

We will end the chapter with a brief discussion of issues which were not mentioned in paper I.

High-resolution level sets will for some applications, such as the Enright test of paper I, result in a small grid cell size, Δx. This is problematic due to the CFL condition which dictates that the level set surface cannot move more than Δx during one time step: Even though we parallelize the computations to reduce the running time, it takes a lot of iterations to advance the solution to a given time t due to the small time steps, Δt. Therefore it does not seem viable to increase the resolution beyond a certain point for some applications when using explicit numerical time integration. Instead, one could devise implicit integration schemes that are not limited to small time steps in order to remain stable.

Recently, solid-state disks have been emerging as a viable storage solution for consumer-level desktop and laptop computers. In terms of size, access speed and pricing it is placed in the memory hierarchy between main memory and regular magnetic disk storage. However, it is still quite expensive compared to magnetic disks and the largest sizes available are still orders of magnitude less. The advance of solid-state disks does not diminish the need to develop out-of-core algorithms since the latency is still some orders of magnitude slower than main memory (typically a couple of microseconds compared to less than a hundred nanoseconds), and since the ongoing trend within computing is to treat larger and larger data sets. Unfortunately, we have not had the opportunity to experiment and compare our new framework with the framework of Nielsen *et al.* [91] in the context of solid-state disks.

CHAPTER 5

FLUID SIMULATION FOR COMPUTER GRAPHICS

Fluids constitute some of the most impressive and fascinating natural phenomena in the world. We have always been attracted to and stood in awe of the mighty turmoil of frothing rivers and waterfalls as well as the treacherous open sea. The wild and complex beauty of these marvels makes understanding them and reproducing them a seemingly daunting task, and throughout time many efforts have been put into doing so. At the other end of the spectrum we find the more subtle appeal of a few drops of dew running down a leaf. Being able to reproduce these effects computationally puts focus on another set of requirements such as correctly handling the surface tension which shapes the drops. Another phenomenon which we commonly do not even regard as a fluid is the air we breathe. In fact, air is at the heart of just as many impressive wonders as water. From large scale phenomena such as clouds or the gasses of volcanic eruptions to the intricate swirling patterns of smoke and fire from a match or a candle, air is the medium for many interesting effects. Being able to do simulations of fluid phenomena such as water and smoke has applications in a variety of fields ranging from engineering disciplines such as CFD where accuracy is paramount to visual effects in films and computer games, where a visually pleasing appearance is sufficient. Many of the methods used in computer graphics have been adapted from CFD and since they are computationally intensive, some simplifying assumptions have been made to make them more feasible for animation.

Before we move on to describing *how* we typically simulate fluids in computer graphics and visual effects, we will briefly discuss a number of reasons *why* it is often necessary to utilize simulations rather than traditional geometric modeling or practical effects when wanting to recreate these phenomena. One of the primary reasons for not modeling smoke or water by hand is that they are both very complex phenomena with many intricate details. Figure 5.1 illustrates the enormous richness in both motion and appearance which would require an infeasible amount of time to shape manually. Indeed, if it were possible to easily sculpt and model water by hand, artists would likely do just that. Of course, in computer games and interactive applications, we *need* to simulate — rather than model — since the user changes the behavior of the fluid by interacting with it. Owing to the scope and often destructive nature of the

Figure 5.1 Real smoke and water phenomena are very complex. **Left:** A volcanic eruption sends steam and various other particles into the air in a highly turbulent manner. **Right:** Multiple crashing waves result in foam, mist and spray.

desired effects, it is prohibitively expensive to perform them as practical effects at full scale. Unfortunately, it is also not possible to use miniatures convincingly since real fluid phenomena have *scale*. Figure 5.2 shows that while some properties such as the overall speed of the fluid can be somewhat overcome by shooting at higher camera speeds to slow the movement down, other properties, such as surface tension, affect *e.g.* the size of droplets which results in a fake look. Using a less viscous fluid such as kerosene or methanol instead of water is typically not an option due to the fire hazard. Finally, real fluid effects are actually really hard to dress and control [105], and — in contrast to simulations — the director only has one chance to shoot and is unable to move the camera around afterwards to obtain the best view.

On the other hand, simulations are not easy to utilize and understand either, and there really is no "silver bullet" method which works great for every fluid effect and every camera shot. Controlling fluid simulations done via physically based simulation techniques is for example hard. Essentially, one is only able to specify the initial conditions and then hope for a satisfying outcome of the often time-consuming simulation. However, the director will often express that he wants the water or smoke to exhibit a certain anthropomorphic characteristic such as being menacing or friendly, serene or agitated. Furthermore, a stylized look and behavior might be desired, and it is often not clear how these requirements translate to the physical parameters of the simulation. To make matters more complicated, sometimes a completely non-physical behavior is required such as the river of horses in "Lord of the Rings: Fellowship of the Ring" or the tar monster from "Scooby Doo 2: Monsters Unleashed". Another example is when a believable physical look is desired, but the exact and correct physical behavior does not behave as wanted. Chapter 1 provided two examples of this situation from "Ratatouille" and "The Day After Tomorrow". Hence, it is not enough to be able to produce a realistic fluid behavior — it must also be *directable*. We will return to this discussion in chapter 7.

This chapter focuses on describing the basics of fluid simulations for computer graphics while the next chapter describes various extensions and improvements. The

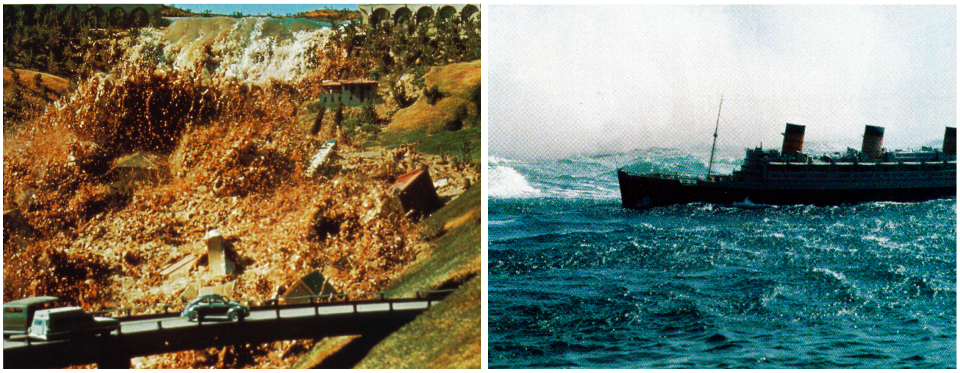


Figure 5.2 Real fluid phenomena have scale which makes it hard to use miniatures convincingly. **Left:** A miniature dam is destroyed in "Earthquake" (1974). ©Universal Pictures. Notice how unwanted droplets form due to surface tension. **Right:** A miniature open sea in "The Poseidon Adventure". ©20th Century Fox. Both images reprinted from [105].

next section will introduce the theory and present the equations which describe the motion of fluids, and in section 5.2 we explain a numerical scheme which solves the equations on a computer. Section 5.3 explains how to use this numerical scheme for simulating fluid phenomena such as smoke and water in practice. Finally, the last section will briefly present an alternative model and method for doing fluid simulations.

5.1 The Equations for Fluid Flow

CFD is a vast field of research and covering it in its entirety does not make sense in the context of this dissertation. Instead, we will focus on what is typically needed to produce convincing fluid behavior for visual consumption. As stated previously, visual appearance is most important, and simplifying as much as possible is always allowed (and encouraged). In some instances, one can get away with a very rough approximation based on *height fields*, such as shallow water simulations which completely ignore vertically varying velocities within the liquid [12]. However, the dynamics of most fluid flow of interest in computer graphics are governed by the (in)famous *incompressible Navier-Stokes equations*, which are a set of PDEs that must hold throughout a three dimensional fluid velocity field $\vec{U}$

$$\frac{\partial \vec{U}}{\partial t} + (\vec{U} \cdot \nabla)\vec{U} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{U} + \vec{F} \tag{5.1}$$
$$\nabla \cdot \vec{U} = 0$$

Here ρ is the density of the fluid, p is pressure, *i.e.* the force per unit area that the fluid exerts on anything, $\nu \nabla^2 \vec{U}$ is the shear stress or internal friction term under the assumption that the fluid is incompressible and Newtonian[1]. ν is the kinematic viscosity which defines how viscous the fluid is, *i.e.* how much it resists deforming while it flows. Finally, $\vec{F}$ is body forces per unit volume, including gravity, buoyancy and other

[1] Note that the ∇^2 operator is the Laplacian which is sometimes denoted by Δ or $\nabla \cdot \nabla$.

forces. Note, that equation 5.1 is actually three equations — one for each component of the velocity field — combined into a single vector equation. Before looking closer at the individual terms of these equations, we will justify their use and simplify them a little.

The primary simplifications typically applied in computer graphics are assuming that the fluid is *incompressible* and *inviscid*. The first assumption has already been applied in the above equations. It is a reasonable assumption given that fluid simulations are typically used for flows with velocities well below the speed of sound which means that the effects of compressibility are negligible. One exception is explosions which are typically handled through ad hoc solutions since modeling a compressible fluid numerically is prohibitively expensive due to strict time step restrictions. The second assumption is also valid when simulating water or smoke as the viscosity of these phenomena is very low under the physical circumstances typically relevant for films or games. In fact, even if viscosity is not explicitly modeled, the numerical methods employed to solve the equations introduce artificial viscosity as explained in section 2.3.4. The equations used to describe the motion of the fluid are simplified due to these assumptions, and if it is further assumed that the fluid has a constant density $\rho = 1$, the fluid can be fully described by a velocity field $\vec{U}$ and a pressure field p. The equations are now called the *inviscid Euler equations*

$$\frac{\partial \vec{U}}{\partial t} + (\vec{U} \cdot \nabla)\vec{U} = -\nabla p + \vec{F} \tag{5.2}$$

$$\nabla \cdot \vec{U} = 0 \tag{5.3}$$

5.1.1 Derivation of the Inviscid Euler Equations

The inviscid Euler equations can be derived from conservation of mass and momentum [12]. Equation 5.3 thus arises from conservation of mass. Consider the mass M of a fluid with density $\rho = 1$ inside an arbitrary, fixed volume Ω. Since mass cannot be created or destroyed inside the volume, the rate of change of mass is given by the total mass flow *out* of the volume

$$\frac{\partial M}{\partial t} = -\iint_{\partial\Omega} \vec{U} \cdot \vec{N}$$

where $\vec{U}$ is the velocity of the fluid as usual and $\vec{N}$ is the outward pointing normal of the boundary $\partial\Omega$. Applying Gauss' theorem (also known as the divergence theorem), we change the above equation to a volume integral

$$\frac{\partial M}{\partial t} = -\iiint_{\Omega} \nabla \cdot \vec{U} \tag{5.4}$$

Because the volume of an incompressible fluid must be constant and we have assumed that $\rho = 1$, the mass of Ω must also remain unchanged

$$\frac{\partial M}{\partial t} = 0$$

Since this must be true for any volume Ω, the integrand of equation 5.4 must itself be zero, and we end up with the *incompressibility condition* of equation 5.3. A velocity field which satisfies this equation is called *divergence-free*.

Equation 5.2 similarly arises from conservation of momentum, and it is in fact Newton's second law, $\vec{F} = m\vec{a}$, in disguise, *i.e.* it relates how the fluid accelerates due to the forces acting on it. To see how it appears, let us look at the fluid as a collection of particles. Each particle represents a small piece of fluid with an associated mass m, volume V and velocity $\vec{U}$. In order to integrate the particles forward in time, we need to find the forces acting on each particle and then use Newton's second law to accelerate them. There are (at least) two forces in play for an inviscid fluid. The first is pressure p, where the intuition is that higher-pressure regions push on regions of lower pressure. Thus, we only see an effect when there is a pressure difference. Since pressure is a contact force, it only acts on the boundary of the volume of our particle. The second force is due to gravity $\vec{G}$ and it acts throughout the volume. Hence, the net force $\vec{F}_{net}$ is given by

$$\vec{F}_{net} = -\iint_{\partial\Omega} p\vec{N} + \iiint_{\Omega} \rho\vec{G}$$

Once again, we can apply Gauss' theorem to convert the surface integral to a volume integral, and we obtain

$$\vec{F}_{net} = \iiint_{\Omega} -\nabla p + \rho\vec{G} \tag{5.5}$$

Recall that we have assumed $\rho = 1$, and we can now recognize the integrand of equation 5.5 as the right-hand side of equation 5.2. They are the forces per volume which act on the fluid. Note, that while we have only considered gravity as an external body force, there can be several other external forces such as artificial buoyancy and vorticity confinement (see chapter 6) in practice.

As we will see in the next section, we are going to solve the inviscid Euler equations using an Eulerian approach (recall chapter 2), so we need to couple the acceleration of the Lagrangian particles to the Eulerian viewpoint. Essentially, we want to answer the question of how the velocity $\vec{U}$ is changing at a fixed point in space as the particles are carried around. The key to connecting these two viewpoints is the material derivative, *i.e.* the derivative taken along a path moving with velocity $\vec{U}$. It is written as $\frac{D}{Dt}$, and the acceleration is given by $\frac{D\vec{U}}{Dt}$. For a generic quantity q the material derivative is defined as

$$\frac{Dq}{Dt} \equiv \frac{\partial q}{\partial t} + \vec{U} \cdot \nabla q$$

The first term, $\frac{\partial q}{\partial t}$, is just the rate of change of q at that fixed point in space — an Eulerian measurement. The second term, $\vec{U} \cdot \nabla q$, expresses how much of that change is due to changes in the fluid flowing past, *e.g.* the temperature changing because cold air is being replaced by hot air, and not because the temperature of any particle is changing. $\frac{Dq}{Dt} = 0$ is called an advection equation, since it states that the quantity q is just moving around but not changing in the Lagrangian viewpoint. We have seen this equation before in chapter 2, where it described the dynamics of a level set. Taking the

material derivative of $\vec{U}$, we see that we obtain the left-hand side of equation 5.2. For a more rigorous derivation of equations 5.2 and 5.3 which handles fluids with varying density, please refer to the book by Bridson [12].

5.1.2 Pressure

When simulating incompressible fluids, we need to make sure that the fluid velocity stays divergence-free. This is where the pressure, p, comes into play. Intuitively, pressure is whatever it takes to keep the fluid at a constant volume, *i.e.* make the fluid velocity field, $\vec{U}$, divergence-free. We can derive an equation which defines what p needs to be, by taking the divergence of both sides of the momentum equation (5.2)

$$\nabla \cdot \frac{\partial \vec{U}}{\partial t} + \nabla \cdot ((\vec{U} \cdot \nabla)\vec{U}) = -\nabla \cdot \nabla p + \nabla \cdot \vec{F}$$

Changing the order of differentiation in the first term yields $\frac{\partial}{\partial t}\nabla \cdot \vec{U}$ which is zero because of the incompressibility condition (5.3). If we rearrange the remaining terms, we get the following equation for pressure which is known as a *Poisson equation*

$$\nabla \cdot \nabla p = \nabla \cdot (-(\vec{U} \cdot \nabla)\vec{U} + \vec{F}) \tag{5.6}$$

5.1.3 Boundary Conditions

So far we have only looked at the interior of the fluid we are simulating and ignored the boundaries. We will now briefly describe two boundary conditions: *solid boundaries* and *free surfaces*[2]. For solid boundaries, the boundary conditions must ensure that fluid does not flow into or out of the solid. Thus the normal component of the fluid velocity has to be zero, *i.e.* $\vec{U} \cdot \vec{N} = 0$, where $\vec{N}$ is the outward pointing normal of the solid boundary. If the solid boundary is moving, the normal component of the fluid velocity should match the normal component of the solid's velocity to ensure that the normal component of the *relative* velocity is zero

$$\vec{U} \cdot \vec{N} = \vec{U}_{solid} \cdot \vec{N}$$

This is a called a *free-slip* condition since the fluid is allowed to slip freely past the solid boundary, and it is the correct, physical solution for incompressible, inviscid fluids. For viscous fluids, one can use a *no-slip* condition, which states that $\vec{U} = \vec{U}_{solid}$. These types of boundary conditions, where we set the values of the solution explicitly, are called *Dirichlet boundary conditions*. The pressure at solid boundaries should enforce the solid wall boundary conditions also (in addition to making $\vec{U}$ divergence-free). Since we subtract the gradient of pressure from the velocity field, we need to make sure that the normal derivative of p is zero: $\frac{\partial p}{\partial \vec{N}} = 0$. This is known as a *Neumann boundary condition*.

The free surface represents the other type of boundary we are interested in. It is essentially where we stop modeling the fluid. For a typical water simulation, it is the

[2]Other important boundaries include the boundary between two different fluids in a two-phase flow simulation, and we refer the interested reader to papers such as that by Hong and Kim [45].

part of the water surface that is not in contact with a solid boundary[3]. We model the outside as a region with constant pressure. We can set the pressure to any arbitrary constant, since only the derivatives of p matter. Thus at a free surface, we have the boundary condition $p = 0$, and we do not modify the velocity in any way. For smoke simulations, the free surface is at the boundary of the simulation domain. Past the boundaries the fluid continues on, but we are not tracking it. We should allow fluid to enter and exit freely so a free surface boundary condition with $p = 0$ seems natural even though there is no visible surface.

5.2 Solving the Inviscid Euler Equations Numerically

Looking at the derivation of the inviscid Euler equations (5.2 and 5.3), it may seem most natural to solve them by applying a Lagrangian particle system, and indeed this has been done in several schemes, including *Smoothed Particle Hydrodynamics (SPH)* [19, 80]. However, we will approach the solution using an Eulerian representation similar to the one we know from level sets since it allows us to reuse numerical approximations to derivatives such as FDs [60] (section 2.3.1 gives an introduction). For clarity, we will restrict our discussion to two dimensions. Rather than trying to solve the complicated equations at once, we apply a numerical method called *operator splitting*, which separates the equations into their component parts and solves each of them separately in turn. This method was originally introduced to computer graphics by Stam [114]. It was presented along with *unconditionally stable* methods for solving the resulting simpler equations, hence it is called the *Stable Fluids* method. The equations account for advection, body forces and incompressibility, respectively, and they are as follows:

$$\frac{\partial \vec{U}}{\partial t} = -(\vec{U} \cdot \nabla)\vec{U} \qquad \text{(advection)} \qquad (5.7)$$

$$\frac{\partial \vec{U}}{\partial t} = \vec{F} \qquad \text{(body forces)} \qquad (5.8)$$

$$\frac{\partial \vec{U}}{\partial t} = -\nabla p$$
$$\text{such that } \nabla \cdot \vec{U} = 0 \qquad \text{(pressure projection / incompressibility)} \qquad (5.9)$$

As mentioned, we solve these equations sequentially using the solution of the previous equation as input to the next. It should be noted that advection should always be performed in a divergence-free velocity field and hence equation 5.7 should always be solved immediately after equation 5.9. More precisely, given numerical schemes `advect` and `project` for solving equations 5.7 and 5.9 separately, the stable fluids method integrates as follows for each time step to obtain $\vec{U}^{n+1}$ from $\vec{U}^n$:

1. $\hat{\vec{U}} = \texttt{advect}(\vec{U}^n, \Delta t)$

[3] There actually *is* another fluid on the other side of the water surface, namely the air, but often we do not bother simulating both fluids.

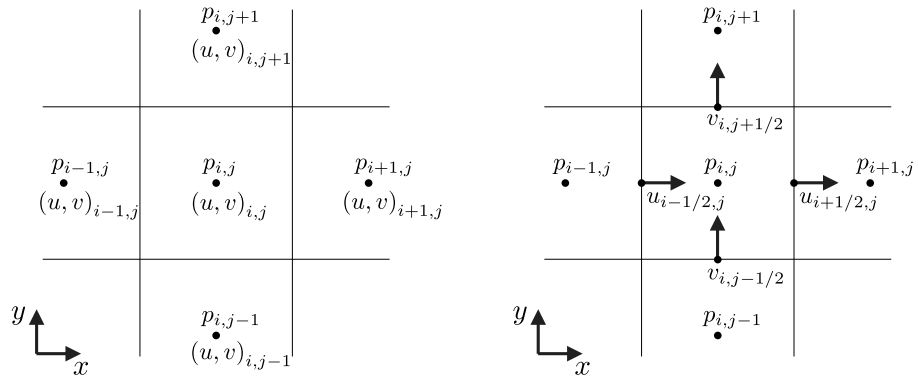


Figure 5.3 Left: A regular grid in two dimensions. The velocities are co-located with the pressure. This type of grid is in many respects simpler to work with. **Right:** A MAC grid in two dimensions. The velocities are split up into their components, which are moved to the faces of the grid cells in a staggered arrangement. This improves second order central FD approximations as explained in the text.

2. $\tilde{\vec{U}} = \hat{\vec{U}} + \Delta t \vec{F}$
3. $\vec{U}^{n+1} = \texttt{project}(\tilde{\vec{U}}, \Delta t)$

Note that equation 5.8 is simply solved using a forward Euler time step. Before we move on to describing the numerical schemes for solving advection (5.7) and pressure projection (5.9), we will briefly look at how we discretize the fluid velocities, $\vec{U} = (u,v,w)$, and pressure, p.

5.2.1 Spatial Discretization

The left part of figure 5.3 shows the two-dimensional grid, which we are familiar with from chapter 2. Velocity and pressure samples are co-located in the center of grid cells. We will reuse the notation from chapter 2 as well and for instance denote the pressure at $(i\Delta x, j\Delta y)$ as p_{ij}. Note that we will assume that the grid is uniformly sampled in all directions, *i.e.* $\Delta x = \Delta y$. We would like to use an unbiased, second order accurate, central FD when for instance discretizing ∇p in equation 5.9

$$p_x^o = \frac{p_{i+1j} - p_{i-1j}}{2\Delta x} \tag{5.10}$$

However, this formula has a major problem since it ignores the pressure p_{ij} at the point where the derivative is taken. Therefore the approximation could evaluate to zero even though p_{ij} is radically different from p_{i-1j} and p_{i+1j}. Technically, formula 5.10 has a *non-trivial null-space*, which means that the set of functions for which the formula evaluates to zero contains more than the constant functions it should be restricted to. When approximating $\nabla \cdot \tilde{\vec{U}}$, *i.e.* the divergence of the fluid velocity field after solving for advection and body forces, we also want to be able to use an unbiased second order accurate FD. However, using the above formula, a highly diverging field, such as $\vec{U}_{ij} = ((-1)^i, (-1)^j)$, would result in zero divergence.

To remedy these problems, Harlow and Welch introduced the *Marker-and-Cell (MAC)* method in the early days of CFD [41]. They proposed a *staggered* MAC grid where the fluid velocity components are sampled on cell *faces* rather than centers. This grid was later introduced to computer graphics by Foster and Metaxas [31]. The right part of figure 5.3 illustrates the staggered grid in two dimensions. Since the velocity components are sampled on cell faces, we refer to them using half indices, such as $u_{i+\frac{1}{2}j}$ for the horizontal velocity sampled on the face between cells (i, j) and $(i+1, j)$. We can now naturally estimate the derivative of *e.g.* u_{ij} using a central FD without skipping any values: $u'_x = \frac{u_{i+\frac{1}{2}j} - u_{i-\frac{1}{2}j}}{\Delta x}$. The MAC grid is thus constructed to work well with the pressure projection operation which solves for incompressibility. However, the downside of this staggered arrangement is that we are no longer able to look up a full velocity vector at any point without performing some kind of interpolation, which makes for instance advection more cumbersome.

The MAC grid generalizes to three dimensions in a straightforward manner.

5.2.2 Semi-Lagrangian Advection

We will now look at how the `advect` scheme solves the advection equation (5.7). Intuition suggests using a simple forward Euler time integration coupled with central FD approximations of the spatial derivatives. Thus, we will for example get the following equation for the u component of the fluid velocity at (i, j) in two dimensions

$$\frac{u^{n+1}_{ij} - u^n_{ij}}{\Delta t} = -u^n_{ij} \frac{u^n_{i+\frac{1}{2}j} - u^n_{i-\frac{1}{2}j}}{\Delta x} - v^n_{ij} \frac{u^n_{ij+\frac{1}{2}} - u^n_{ij-\frac{1}{2}}}{\Delta y}$$

Rearranging gives us a formula for u^{n+1}_{ij}. Unfortunately, it turns out that forward Euler coupled with this central FD scheme is *unconditionally unstable*, *i.e.* no matter how small Δt is chosen, the numerical solution will always eventually blow up. Rather than try to repair the problem by applying an upwind scheme or a more sophisticated FD, we will instead apply an unconditionally stable method known as *semi-Lagrangian advection*. Recall that solving the advection equation in a Lagrangian frame is trivial since it is solved simply by moving the particles through the velocity field. Thus, to answer the Eulerian question of getting the new value of some quantity q at some fixed point $\mathbf{x}$ in space, we just need to find the particle that ends up at $\mathbf{x}$ and look up its value of q. This is the central idea behind semi-Lagrangian advection. Specifically, to find the value of $q^{n+1}_{\mathbf{x}}$ we trace an imaginary particle from $\mathbf{x}$ backwards in time through the velocity field and end up at $\mathbf{x}_s$. We then copy the value of q found at that point: $q^{n+1}_{\mathbf{x}} = q^n_{\mathbf{x}_s}$. The tracing itself can be done using a forward Euler integration or a higher order *Runge-Kutta* method [60]. Typically, the imaginary particle ends up somewhere between the values of the grid. Therefore, we need to interpolate the result from nearby values using *e.g.* bilinear interpolation (trilinear interpolation in three dimensions). In the next chapter we will look at ways of improving the numerical accuracy of this basic scheme.

Since the semi-Lagrangian scheme is unconditionally stable[4], there is no strict

[4] We only ever obtain new values through a convex combination of the old values, hence the new

limit on the time step size Δt. However, visual artifacts can occur if Δt is too large, and it is generally a good idea to break the tracing of trajectories up into smaller substeps $\Delta t'$ such that each substep only traces roughly one grid cell, *i.e.* $\Delta t' < \frac{\Delta x}{|u|}$ in one dimension.

5.2.3 Ensuring Incompressibility

We finish our tour through the Stable Fluids solver by looking at the `project` scheme for solving the pressure projection (5.9). We start by using a forward Euler time integration to obtain an equation for the new velocity field $\vec{U}^{n+1}$

$$\vec{U}^{n+1} = \tilde{\vec{U}} - \Delta t \nabla p \tag{5.11}$$

where $\tilde{\vec{U}}$ is the diverging, intermediate velocity field resulting from the two first steps of the Stable Fluids solver. We need to determine the pressure p such that the incompressibility condition is satisfied as well as the solid and free surface boundary conditions. The Poisson equation (5.6) gives us exactly this. We restate it here in a slightly different form, which is the one we will solve

$$\Delta t \nabla \cdot \nabla p = \nabla \cdot \tilde{\vec{U}} \tag{5.12}$$

Equation 5.11 is called the *pressure update*, and we will start by discretizing it using a central FD approximation of the gradient of p

$$u^{n+1}_{i+\frac{1}{2}j} = \tilde{u}_{i+\frac{1}{2}j} - \Delta t \frac{p_{i+1j} - p_{ij}}{\Delta x}$$
$$v^{n+1}_{ij+\frac{1}{2}} = \tilde{v}_{ij+\frac{1}{2}} - \Delta t \frac{p_{ij+1} - p_{ij}}{\Delta y}$$

Similarly, we discretize the divergence of $\tilde{\vec{U}}$ on the right-hand side of equation 5.12

$$(\nabla \cdot \tilde{\vec{U}})_{ij} \approx \frac{\tilde{u}_{i+\frac{1}{2}j} - \tilde{u}_{i-\frac{1}{2}j}}{\Delta x} + \frac{\tilde{v}_{ij+\frac{1}{2}} - \tilde{v}_{ij-\frac{1}{2}}}{\Delta y} \tag{5.13}$$

The pressure update is applied on the faces of the grid cells while the divergence is the right-hand side of an equation for pressure which is defined at grid cell centers. Everything lines up nicely, and now we can really appreciate the properties of the staggered MAC grid. Finally, we discretize the left-hand side of equation 5.12

$$\begin{aligned}\Delta t \nabla \cdot \nabla p &= \Delta t \left(\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \right) \approx \Delta t \left(\frac{p_{i+1j} - 2p_{ij} + p_{i-1j}}{\Delta x^2} + \frac{p_{ij+1} - 2p_{ij} + p_{ij-1}}{\Delta y^2} \right) \\ &= -\frac{\Delta t}{\Delta x^2} (4p_{ij} - p_{i+1j} - p_{i-1j} - p_{ij+1} - p_{ij-1})\end{aligned} \tag{5.14}$$

where we have used a second order accurate central FD for the second order partial derivatives. We can now combine expressions 5.13 and 5.14 to form a numerical approximation to equation 5.12. We end up with a system of linear equations $Ap = b$

values are bounded by the old ones.

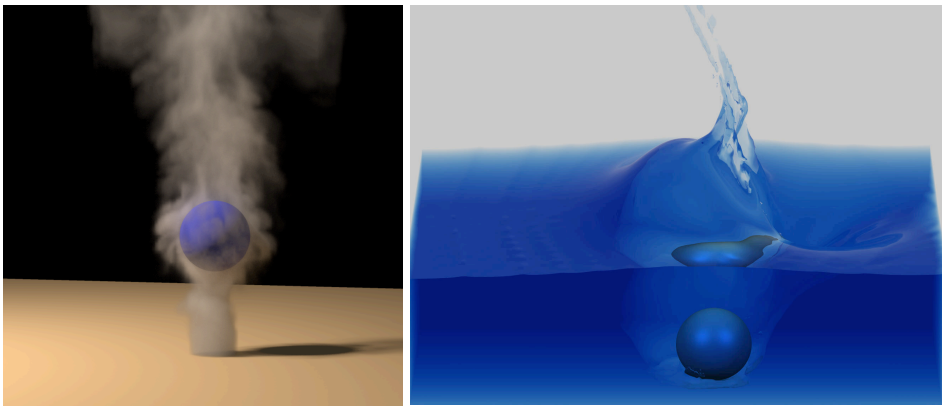


Figure 5.4 Simulated fluid phenomena. **Left:** A column of smoke flows around a solid sphere boundary. Reprinted from [26]. **Right:** A solid sphere boundary is dropped into the water causing a splash. Reprinted from [30].

where p are the pressure unknowns, A is the coefficient matrix with the coefficients from expression 5.14, and b is the divergence in each fluid cell given by expression 5.13. Once the system of equations has been set up[5], it is traditionally solved with a numerical scheme known as the *Preconditioned Conjugate Gradient (PCG)* method along with an *Incomplete Cholesky Preconditioner* [60]. Further details on implementing a fluid solver, including the PCG method and correct handling of boundary conditions in the pressure projection, can be found in the book by Bridson [12].

5.3 Simulating Smoke and Water

In the previous sections, we explained all the steps needed to integrate the motion of a fluid through time. There are only a few extra steps required to simulate phenomena such as smoke or water [26, 31, 32], and we will briefly explain these in turn. Figure 5.4 illustrates some of the results achievable using these methods.

For smoke, one popular option is to represent the densities of soot particles s as a scalar field where the values are stored at cell centers. Other quantities, such as temperature T, can be represented in the same way. These quantities are then advected around with the fluid

$$\frac{\partial q}{\partial t} + (\vec{U} \cdot \nabla) q = 0 \tag{5.15}$$

in a separate step using semi-Lagrangian advection just as for the fluid velocity components themselves. New smoke can be added and temperature changes performed in a separate step similar to how equation 5.8 is solved. Finally, the density of the smoke and the temperature changes can be used to create an effect similar to buoyancy by influencing the fluid velocity field through an acceleration

$$\vec{B} = (\alpha s - \beta(T - T_{ambient}))\,\vec{G}$$

[5] For numerical reasons, the Poisson equation is negated to make the involved system positive definite.

where α and β are non-negative control coefficients, and $T_{ambient}$ is the ambient temperature.

For water, we need to track the water surface. As mentioned in previous chapters, this is often done using the level set method where the level set surface is advected using the fluid velocities. The level set on the other hand defines the free surface which is used when setting up boundary conditions in the fluid simulator. It should be noted that the water velocities must be extrapolated into the air in a band as thick as the narrow band containing the level set in order to ensure correct advection of the surface. One method for doing this involves solving $\frac{\partial q}{\partial t} = -\nabla\phi \cdot \nabla q$ to steady state outside the fluid. Since this dissertation primarily focuses on smoke animations, we will leave the discussion of water simulations at that.

5.4 An Alternative Method

There are a number of other methods for doing fluid simulations which are not based on solving the Navier-Stokes equations using an approach similar to the one described in the previous sections. One of the most interesting for computer graphics is the *Lattice-Boltzmann method (LBM)* [15], which follows the approach of cellular automata to model complex systems using a set of simple, local rules for each cell. It computes the macroscopic quantities of the fluid such as velocity and pressure by simulating the microscopic particles. Recently, this method has garnered a lot of interest in the computer graphics community due to its local, parallel nature [121, 122, 124]. All computations are the same for each grid cell and they only depend on the neighboring cells, which makes the method perfect for a parallel architecture such as modern graphics cards. However, the basic method is limited by strict time step restrictions in order to ensure stability and it also has fairly high memory requirements.

CHAPTER 6

FLUID SIMULATION EXTENSIONS

In this chapter, we will present a number of important improvements that have been proposed for the original Stable Fluids method of Stam [114]. While the Stable Fluids method has a lot of advantageous properties including being unconditionally stable (*i.e.* no matter how large a time step one utilizes, the solution does not grow exponentially) and fairly simple to implement, it suffers from one major disadvantage. It is only first order accurate in space and time, and the amount of numerical dissipation is very high, which unfortunately results in a dampening and smoothing of high-frequency detail and therefore in a less visually interesting flow. Another disadvantage is that solid boundaries are sampled on the same grid as the other quantities of the Stable Fluids method. However, the boundaries may in fact be curved and not lined up with the grid, and this can cause some noticeable visual artifacts near them. The improvements presented in the following sections attempt to remedy these issues.

6.1 VORTICITY CONFINEMENT AND VORTEX PARTICLES

Vortices create many of the interesting swirling, turbulent motions in for instance smoke. Unfortunately, they tend to dissipate quickly during numerical simulation due to numerical dissipation and the operator splitting method. We would like to detect these naturally occurring vortices and somehow enhance them to make the result more visually interesting. The curl operator measures how much a velocity field is rotating around any point. It is defined as $\nabla \times \vec{U}$, which is just the cross product of the differential operator $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$ and $\vec{U}$. In three dimensions it results in a vector, the length of which measures how fast the velocity is rotating, while the direction gives us the axis of rotation. In fact, curl is twice the local angular velocity and we define vorticity $\vec{\omega}$ to be the curl

$$\vec{\omega} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right)$$

Steinhoff and Underhill [116] proposed a technique for preserving vorticity called *vorticity confinement*. The method introduces an extra term to the momentum equation (5.2) which attempts to preserve vorticity. The central idea, which was later introduced to computer graphics by Fedkiw *et al.* [26], is to identify vortices as peaks in the vorticity field and then add a body force to boost the rotational motion around each vortex. The normalized gradient $\vec{N}$ of $|\vec{\omega}|$ will give us unit vectors pointing towards the peaks of vorticity, *i.e.* the centers of rotation of the vortices

$$\vec{N} = \frac{\nabla|\vec{\omega}|}{|\nabla|\vec{\omega}||}$$

With $\vec{N}$ pointing towards the center of rotation, and $\vec{\omega}$ pointing along the axis of rotation, we can obtain a force, $\vec{F}_{conf}$, which enhances the rotation by taking the cross product of these two vector fields

$$\vec{F}_{conf} = \varepsilon \Delta x (\vec{N} \times \vec{\omega})$$

where Δx is introduced to ensure that the force diminishes as $\Delta x \to 0$, and ε is a parameter that controls the magnitude of the vorticity confinement force. Ideally, ε should be somehow connected to the amount of vorticity dissipation. However, to the best of our knowledge, no one has proposed such a coupling, and choosing a suitable ε can be hard. For instance, choosing ε too high may result in unnaturally looking motion, where the vorticity completely overtakes the flow resulting in a random turbulent chaos which makes the smoke look "alive". It is also unfortunate that ε is the same everywhere in the simulation domain.

Selle *et al.* [108] improve upon these problems by introducing the *vortex particle method*. The idea is to add Lagrangian particles carrying dynamic vorticity, *i.e.* each particle has an associated vector $\vec{\omega}$. This vorticity is then transferred from the particles back to the Eulerian velocity field using Gaussian kernels. This ensures that vorticity confinement is only applied locally. The correct motion of the particles can be derived by applying the curl operator to both sides of the momentum equation (5.2) to put it into *vorticity form* [12]. This results in two terms: vorticity advection and vorticity *stretching*. The vorticity advection is handled simply by advecting the particles themselves using a higher order Runge-Kutta scheme [60]. The vorticity is stretched by solving the equation $\frac{\partial \vec{\omega}}{\partial t} = (\vec{\omega} \cdot \nabla)\vec{U}$ using a simple forward Euler integration and FDs. This method allows highly turbulent flows without degenerating to turbulent chaos, and it is well-suited for creating the rolling smoke that is characteristic of many explosions. However, adding particles remain an ad hoc undertaking in terms of how many particles should be used and when to add them.

6.2 Reducing Numerical Dissipation of Advection

Several methods which can help reduce the numerical dissipation of the semi-Lagrangian approach have been proposed. One such method originates from the field of level sets, but is general enough to be used in the context of semi-Lagrangian advection of fluid velocities also. It is called a *Back and Forth Error Compensation and Correction*

(BFECC) method, and it was proposed by Dupont and Liu [21]. Let $\mathscr{L}_{\Delta x}$ denote a numerical operator which solves the advection equation (5.7) forward in time one time step. Similarly, let $\mathscr{L}_{\Delta x}^{-1}$ denote a numerical operator which solves the advection equation backwards in time. The BFECC method then proceeds as follows for integrating a quantity q forwards one time step:

1. Solve forward in time: $\tilde{q}^{n+1} = \mathscr{L}_{\Delta x}(q^n)$.
2. Solve backward in time: $q_1^n = \mathscr{L}_{\Delta x}^{-1}(\tilde{q}^{n+1})$.
3. Let $q_2^n = q^n + \frac{1}{2}(q^n - q_1^n)$.
4. Solve forward in time: $q^{n+1} = \mathscr{L}_{\Delta x}(q_2^n)$.

In our case, $\mathscr{L}_{\Delta x}$ is semi-Lagrangian advection and q is the fluid velocity components. For level sets, $\mathscr{L}_{\Delta x}$ could be a third order TVD RK [112] coupled with HJ WENO [69], while q would be the signed distance function, ϕ. The numerical error is assumed to be roughly the same for both numerical operators, and hence we can estimate the error by $e = \frac{1}{2}(q^n - q_1^n)$. We then add e to q^n in order to remove the principal components of the error from q^{n+1}. This can result in an improvement of the order of the numerical method both temporally and spatially [21].

If the advected velocity field has details varying at the size of grid cells, Δx, or smaller, resampling will destroy them or cause them to alias as lower-frequency artifacts. In fact, the Nyquist sampling limit dictates that the maximum spatial frequency that can be reliably advected has period $4\Delta x$, which is even worse. Lagrangian particles, on the other hand, do not lose information as no resampling is performed. As mentioned in the previous chapter, this has lead to methods based entirely on particles such as SPH [19, 80]. However, Eulerian grids are more efficient and accurate when enforcing incompressibility, and thus *hybrid particle* methods which seek to utilize the best properties of both approaches have emerged. The *Particle-in-Cell (PIC)* method by Harlow and Welch [41] replaces the semi-Lagrangian step with actual particle advection, while integrating all other terms on the grid. In particular, the advection step first transfers the quantity to be advected, *e.g.* the fluid velocity components, onto a number of particles for each grid cell using interpolation. It then advects the particles using a higher order Runge-Kutta method. Finally, it transfers the quantity back to the grid from the particles using a weighted average. Unfortunately, the method suffers from excessive numerical dissipation because quantities are averaged from particles to the grid, and then these *smoothed* quantities are interpolated back to the particles in the next step, thus compounding the smoothing.

The *fluid implicit particle (FLIP)* method introduced to incompressible flow by Zhu and Bridson [136] removes the numerical dissipation of basic PIC using a simple variation on the sceme: Rather than interpolating a quantity back to the particles, the *change* in the quantity (resulting from body forces and pressure projection computed on the grid) is interpolated and *added* to the particle value rather than replacing it. Therefore, smoothing is not accumulated and FLIP is virtually free of numerical dissipation. The method can introduce some noise, however, and therefore it can be a good idea to *blend* it with a regular PIC update. Both PIC and FLIP are first order

accurate. This appears to be adequate for advecting *e.g.* velocities, smoke densities, and temperatures.

Finally, we note that the numerical dissipation of the semi-Lagrangian approach due to interpolation can be decreased by using the monotonic cubic interpolation scheme of Fedkiw *et al.* [26]. However, care must be taken to ensure that interpolated values are bounded by the values of the involved grid points. Otherwise, the semi-Lagrangian method loses the property of being unconditionally stable.

6.3 More Accurate Boundaries

The traditional Stable Fluids approach simply labels grid cells as either being solid, fluid or air in preparation of the boundary conditions for advection and pressure projection. Thereby it reduces the geometry of solid boundaries to voxelized, sampled representations. For an inclined or curved solid boundary this can lead to *stair step* artifacts where the liquid rests on the boundary at regular intervals rather than smoothly flowing down it. One interesting alternative is to replace the MAC grid with an unstructured tetrahedral mesh [28]. This allows for a perfect alignment of solid boundaries with the simulation grid. However, it also comes with the overhead of constructing and simulating on an unstructured mesh which is more complicated than working with a fixed Eulerian grid.

Another solution can be found in the *finite volume* method. The method discretizes the integral form of the incompressibility condition $\iint_{\partial C} \vec{U} \cdot \vec{N} = 0$, where C is a *control volume*. In the interior of the fluid, each grid cell is simply used as C, and the boundary integral is approximated by summing the area of each cell face (Δx^2) multiplied with the velocity component stored at the face center. Near solid boundaries, we instead use just the fluid part of the grid cell as the control volume. This means that the area of some faces will be reduced to the fraction of the face which lies within the fluid. Ultimately, we end up with a system of linear equations for pressure, which is similar to the usual system, and can be solved with the same numerical methods. The only difference is that pressure values immediately within solid boundaries are a part of the system now in order to allow for a more accurate solution. Since this method uses pressures inside solids, it can not handle solids which are thinner than one grid cell. In these cases, it may be preferable to fall back to the usual voxelized treatment of solids. Guendelman *et al.* [40] demonstrate how to do this and we refer to their paper for further details.

Recently, Batty *et al.* [6] introduced an approach based on calculus of variations which uses fluid volume fractions in the discretization. The pressure projection operation is formulated as a constrained minimization problem, and they show that the discretization of this formulation leads to boundary conditions which are free of the previously mentioned stair step artifacts. We will look closer at variational calculus in the next chapter. For more information, we refer to the original paper and the book by Bridson [12].

CHAPTER 7

CONTROLLING FLUID SIMULATIONS

Chapter 5 presented the Stable Fluids method [114] for simulating fluids in computer graphics. It furthermore motivated the need for actually *simulating* these phenomena rather than modeling them by hand or using practical effects. While the methods presented in chapter 6 improved several of the numerical issues of the original method, we have so far ignored the important problem of controlling the final visual appearance and behavior of a fluid simulation. The work presented in this chapter and papers II and III targets this problem of making fluid simulations more easily controllable and directable[1]. As discussed in the beginning of chapter 5, one of the central problems in producing controllable and directable behavior using a fluid simulator lies in the fact that we are solving a so-called *initial value problem*: We specify the initial conditions and wait for the outcome of the computationally expensive simulation. Unfortunately, the desired behavior and appearance can be hard to express in these initial conditions. Furthermore, the non-linear, unstable nature of the inviscid Euler equations means that tweaking the parameters using a previous result as a frame of reference often results in a completely different, unexpected behavior. This means that a lengthy trial and error process is often involved in art-directing a certain fluid simulation in visual effects. Since fluid simulations are computationally expensive, artists often employ the workflow illustrated to the left in figure 7.1, which we will call the *regular* design cycle. They simulate on a coarse, low-resolution grid in order to obtain a reasonable turnaround time when refining their parameters. Once they have a satisfactory result, they increase the resolution of the simulation to add more detail. Unfortunately, this often completely changes the overall behavior and visual impression of the animation, which may cause the composition to fail from the director's viewpoint [36]. This property is especially evident when the initial grid resolution is very low and the solution has far from converged primarily due to the numerical viscosity of the discretization (and, again, the non-linearity of the inviscid Euler equations).

Our proposed fluid control framework is currently targeted at smoke simulations and assumes that the regular design cycle illustrated in figure 7.1 is possible and fea-

[1] Some of the text from the papers will be restated in this chapter in order to keep the reading experience coherent.

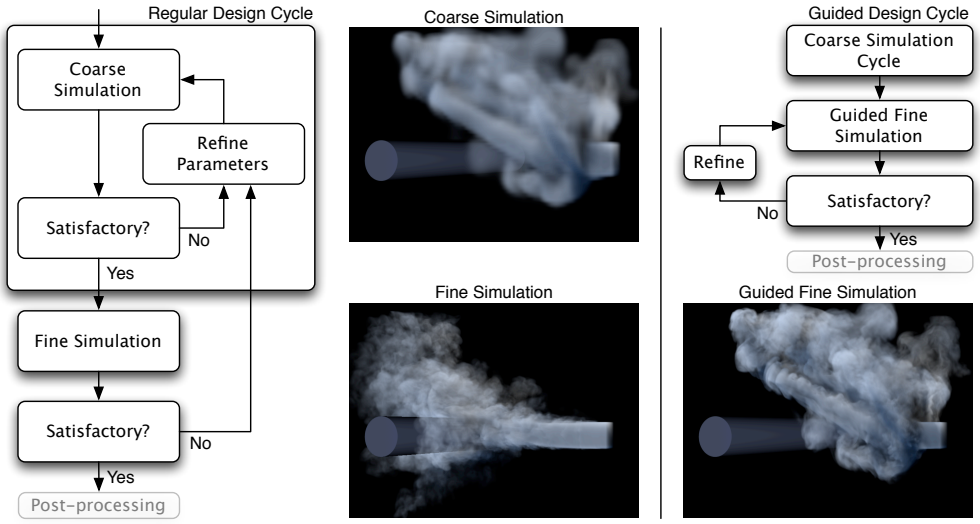


Figure 7.1 Left: A regular design cycle for art-directing a fluid simulation in visual effects. The fine simulation unfortunately looks and behaves completely differently from the coarse simulation. **Right:** The guided design cycle made possible by our fluid control framework. The guided fine simulation maintains the bulk flow of the coarse simulation while allowing higher-frequency detail to develop. The fine simulation results could be fed into a post-processing scheme to further enhance the level of detail.

sible for the given simulation problem. In particular, we assume that it is possible to obtain the desired bulk motion in the coarse simulation. If so, we can apply the *guided* design cycle illustrated to the right in figure 7.1. Once the coarse simulation cycle has provided a satisfying flow, the guided high-resolution simulation is run using our method, which results in an overall motion and behavior similar to the coarse simulation, but with added high-frequency details. This allows for a workflow where the artist can concentrate on experimenting with the parameters in the high-resolution simulation without worrying about changing the approved bulk motion from the low-resolution starting point. The method modifies the usual pressure projection step of a Stable Fluids solver, but is able to reuse all other steps, making it well-suited for integration into an existing fluid simulation pipeline. We rely on *calculus of variations* [63] for making a coupling between the velocity field of a low-resolution simulation and the low frequencies of the velocity field of a high-resolution simulation. Specifically, we derive our mathematical model from a variational formulation of the pressure projection, *i.e.* the pressure projection phrased as a constrained minimization problem. We use *lowpass filters* to ensure that only the low frequencies are being guided, while the higher frequencies are free to develop the desired details. Our model includes *guiding weights* that allow us to specify the guiding strength for each point in the simulation domain. We employ a customized *multigrid* method [14] for efficiently solving the resulting system of linear equations. In total, our method requires less than roughly twice the computation time of a regular simulation of the same resolution.

The remaining sections of this chapter are organized as follows. Section 7.1 highlights a number of other methodologies for providing control over fluid simulations and adding high-frequency detail. It should be stressed that these alternatives are not

precluded by our work, and they can in most cases be combined with our method in order to produce high-quality simulations quickly. Next, section 7.2 briefly introduces calculus of variations and demonstrates how it can be used to derive the Poisson equation from a constrained minimization problem. We then summarize the contributions of papers II and III in the following section. Section 7.4 provides a few extra implementation details, particularly regarding the multigrid solver. They are part of our planned journal paper on fluid control. Finally, the last section discusses the results obtained.

7.1 Control Methodologies

Given the importance of fluid simulations in computer graphics and visual effects in particular, a lot of work has been and is still being put into providing various "knobs" and "handles" for controlling the outcome. These approaches vary in their specific goals and some are specialized for providing a certain effect while others are more generally applicable. Most of the interesting and relevant methods in the context of our control framework have been covered to some extent in paper II, and we will focus here on how they complement our method.

Divergence control is an example of a simple, but effective control mechanism. It was used by Feldman *et al.* [27] as a method well-suited for creating explosion-like motions for fire and smoke simulations. The idea is to modify the right-hand side of the incompressibility condition (5.3) by adding a term, d, which dictates the fractional volume change for each point in the fluid, thus essentially forcing the fluid to diverge when and where we desire it. Setting d to some positive (constant or time-dependent) value inside smoke sources can create more interesting, billowing motions. Similarly, setting d to a negative value somewhere can cause smoke to be sucked towards that area. The modification of the incompressibility condition results in a slight change in the final linear system of our method, but it should be trivial to implement.

Enright *et al.* [24] suggest providing control for water simulations through manipulations of the velocity extrapolation across the free surface. The idea is to simply perform a convex combination of the extrapolated velocities with a number of procedurally or artistically generated velocity fields in order to affect the advection of the free surface represented by a particle level set. They suggest using a wind field in order to make the water look more rough and windblown, or employing the zero velocity field to make the water settle quicker.

Treuille *et al.* [126] propose a scheme for controlling smoke simulations through keyframes specified by the user. It is an optimization problem in that they formulate an objective function which corresponds to how well a simulation matches the specified keyframes. Then they solve for the forces which minimize that function using a gradient descent based optimizer resulting in the smoke assuming the key-framed poses. Later, McNamara *et al.* [75] improved the speed of this approach by employing the adjoint method for computing derivatives used in the optimization process. Since the adjoint method requires computation of adjoint derivatives for each step of the simulation, we would need to compute adjoint derivatives for our modified pressure

projection step in order to combine our method with theirs in high resolution. The adjoint state of pressure projection involves solving the transpose linear system. The paper of McNamara *et al.* exploits that the original pressure projection results in a symmetric system, and hence the adjoint state is computed in the same way as the original state. However, our method results in an asymmetric system, which means that we would have to solve a different system when computing the adjoint. Alternatively, one could apply the adjoint method exclusively to obtain the low-resolution prototype and then use our method for the final high-resolution result.

A class of control methods more suited for immediate combination with our method is the force based control schemes. An example is the work of Fattal and Lischinski [25], which adds a force term to the momentum equation (5.2) designed for driving the smoke towards a given target shape. Additionally, a smoke gathering term is added to the usual equation for smoke density advection (5.15) in order to combat numerical diffusion and allow sharp features in the target shape. Both terms are explicitly defined by the current state of the simulation, and therefore there is no need to perform an optimization to find them. The drawback is that there is no way to directly control how well a target shape is approximated at a given time. Furthermore, the driving force and gathering term must be used carefully in order to preserve a natural looking, rich fluid motion. All the force based control methods are easily combined with our control framework since we only modify the pressure projection step. However, note that in many cases the forces must be added before advecting the velocity field as their effect is not guaranteed to survive the pressure projection step.

Recently, and simultaneously with the development of our control framework, procedural methods for synthesizing the high-frequency detail as a *post-processing* step have emerged. They are typically given a simulation result of fairly low resolution as input and add the higher frequencies procedurally rather than through simulation, thus avoiding the costly running times and memory requirements of a full simulation at high resolution. One example is the *wavelet turbulence* method of Kim *et al.* [58], which uses the wavelet decomposition of a low-resolution simulation to locate missing high-frequency components. These components are then synthesized and added back in a time-coherent manner. One important disadvantage is that this and similar methods can cause otherwise stable, laminar flows to break up into turbulence. The chapter will end with a more thorough comparison of our method to the approach based on wavelet turbulence. As indicated by the semi-transparent *post-processing* box of figure 7.1, the output of our method could be used as input to a post-processing scheme for adding even more detail.

For a discussion of other related control methods, please refer to papers II and III.

7.2 Calculus of Variations

Extremum problems are ubiquitous within a lot of scientific fields, such as mechanics, physics, mathematics, and computer graphics and vision. They involve finding the largest or smallest possible value of a quantity, and they are applicable to a wide range of problems, such as finding the highest peak of a mountain or the shortest path

between two points. The *calculus of variations* can be used to solve problems of this kind. Mathematically speaking, the calculus of variations was developed for solving the problem of minimizing a definite integral. However, we will introduce the intuition behind it looking at the minimization of a function. Say we have the height of a mountain range represented by the continuous and differentiable function $f(x,y)$. At the bottom of a valley all points of an infinitesimal neighborhood must have the same height, *i.e.* at a (local or global) extremum, the rate of change of f must be zero in every possible direction. This is not quite enough to classify extrema in general as it could be a saddle point rather than an extremum. We say that f has a *stationary value* at points where the rate of change is zero. Conveniently, many problems of motion within mechanics require only the stationary value of a certain integral. This includes the variational formulation of pressure projection within fluid mechanics.

In order to mathematically define our exploration of the infinitesimal neighborhood, we introduce the concept of a *variation*. A variation means an infinitesimal, *virtual* displacement. Think of it as a kind mathematical experiment, where we explore what happens with a function when we change the position slightly in an arbitrary way. We write these infinitesimal, virtual changes of our position as $\delta x_1, \delta x_2, \ldots, \delta x_n$ for a function of n variables. The resulting change of the function is called the variation of the function. The first variation of the function $f(x_1, x_2, \ldots, x_n)$ is given by

$$\delta f = \frac{\partial f}{\partial x_1}\delta x_1 + \frac{\partial f}{\partial x_2}\delta x_2 + \cdots + \frac{\partial f}{\partial x_n}\delta x_n$$

If f is to have a stationary value, the first variation must vanish. Since the displacements δx_i are arbitrary, it follows that f has a stationary value if and only if

$$\frac{\partial f}{\partial x_i} = 0, \quad (i = 0, \ldots, n) \tag{7.1}$$

Say we wish to minimize a function subject to some auxiliary constraint $g(x_1, \ldots, x_n) = 0$. This implies that our variables can no longer be varied freely, since they are not completely independent. Therefore, having the first variation vanish does not imply equation 7.1, and we no longer have a *free* variation problem. However, it turns out that by introducing an undetermined factor λ, the problem can be turned back into a free variation problem. Rather than setting the first variation of f equal to zero, we instead look at the first variation of the function $\bar{f} = f + \lambda g$, and treat it as a free variation problem. This method is called the method of *Lagrange multipliers* and it generalizes to an arbitrary number of constraints.

Let's look at how these ideas generalize to finding stationary values for definite integrals. In one dimension, we are given a functional $F(y, y', x)$ of three variables. We are further given the definite integral $I = \int_a^b F(y, y', x)dx$ along with boundary conditions $f(a) = \alpha$ and $f(b) = \beta$. The problem is then to find the function $y = f(x)$ which minimizes I or at least gives it a stationary value. It may appear that this problem is radically different from what we have just considered. Instead of minimizing a function, we are now minimizing a definite integral, and our unknown is now a function, $y = f(x)$, rather than a set of variables, $x_1, \ldots, x_n$. However, it turns out that by replacing the definite integral with a finite sum and using difference coefficients for the

derivate, we can turn the problem back into a problem of minimizing a function of a set of variables. As we carry this approach to the limit, *i.e.* let the discretization interval, Δx, tend to zero, we end up with the so-called *Euler-Lagrange* differential equation

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\frac{\partial F}{\partial y'} = 0$$

which is a necessary and sufficient condition of a stationary value for the definite integral, I, *i.e.* the solution of the equation is a function y which results in I being stationary. The differential equation generalizes to functions of several variables in a straightforward manner.

The above exposition is very condensed, and the interested reader can find further information in the book by Lanczos [63]. We will now briefly demonstrate how calculus of variations can be used in the context of fluid mechanics. Recall that the Poisson equation

$$\nabla \cdot \nabla p = \nabla \cdot \tilde{\vec{U}} \tag{7.2}$$

is used to enforce the incompressibility condition of the inviscid Euler equations. It can be derived from the minimization of the difference between two velocity fields $\vec{U}$ and $\tilde{\vec{U}}$ subject to the constraint that $\vec{U}$ be divergence-free (see [29, pp. 202-204]). Mathematically formulated, one minimizes

$$R = \frac{1}{2}\iiint_{\Omega}\left[\vec{U}(\mathbf{r}) - \tilde{\vec{U}}(\mathbf{r})\right]^2 \tag{7.3}$$

subject to the constraint

$$\nabla \cdot \vec{U}(\mathbf{r}) = 0 \tag{7.4}$$

where $\mathbf{r}$ is the position vector, Ω is the domain, and we have assumed that the fluid density is constant. Employing the method of Lagrange multipliers, the constrained minimization problem (equations 7.3 and 7.4) can be transformed into the unconstrained problem

$$\bar{R} = \frac{1}{2}\iiint_{\Omega}\left[\vec{U}(\mathbf{r}) - \tilde{\vec{U}}(\mathbf{r})\right]^2 - \iiint_{\Omega}\lambda(\mathbf{r})\nabla \cdot \vec{U}(\mathbf{r})$$

where $\lambda(\mathbf{r})$ are the scalar-valued Lagrange multipliers. Taking the first variation, $\delta\bar{R}$, and solving for a stationary point, one obtains the Poisson equation (7.2) and the familiar velocity field correction formula $\vec{U}(\mathbf{r}) = \tilde{\vec{U}}(\mathbf{r}) - \nabla\lambda(\mathbf{r})$ subject to the boundary conditions $\lambda = 0$ on parts of the boundary where outflow is allowed (*i.e.* free surfaces where boundary conditions for $\vec{U}$ are not prescribed) and $\nabla\lambda = 0$ on parts of the boundary where values for $\vec{U}$ are prescribed (*i.e.* solid boundary conditions). Note that this allows for a one-way coupling from moving solid boundaries to the fluid, and that the pressure, p, is in fact the Lagrange multipliers, λ.

7.3 Contributions

As mentioned in the beginning of the chapter, the primary goal of our fluid control framework is to facilitate a workflow, wherein the artist can *reliably* use coarse, prototype simulations for parameter tuning. The bulk motion of the prototype should be

retained as well as possible when increasing the resolution of the simulation to produce a final, more detailed result. Our framework only modifies the usual pressure projection step of a Stable Fluids solver, and is able to reuse all other steps, making it well-suited for integration into an existing fluid simulation pipeline. In particular, the framework does not disrupt the remaining workflow of hardware, software and artists.

To summarize, the work presented in paper II makes the following contributions:

- A mathematical model for guiding a high-resolution velocity field with a low-resolution velocity field based on calculus of variations. We also provide the resulting set of linear equations.
- A practical implementation of our guiding framework, including methods for lowpass filter estimation and handling of boundaries. In particular, to handle boundaries, we adopt the *penalization method* [4] and provide an implicit discretization to ensure numerical stability.
- A custom multi-threaded multigrid implementation based on a fast and compact dynamic matrix storage format. Our multigrid solver maintains a staggered grid on each level which improves the convergence ratio by a factor of two. As a result, a guided simulation requires less than twice the simulation time of a regular simulation of the same resolution.
- Early investigations of the parameters of our framework, such as the strength of the guiding weights and the size of the lowpass filters.

Paper III improves upon the previous paper and contributes the following:

- A novel mathematical model that leads to more efficient space- and time-dependent guiding of smoke animations. The efficiency is derived from an equation system, where the matrix does not have to be recomputed when the guiding weights change over time. Furthermore, the computation times and storage requirements are reduced compared to the previous paper when using space-dependent guiding weights.
- Exploration of time-dependent guiding for artistic effects and for increasing the amount of high frequency features in the high-resolution guided flows. We propose guiding weights based on the smoke densities and well as guiding weights based on error estimates. Additionally, we explore using artistically animated guiding weights.

The unfinished journal paper will provide more detailed descriptions of all steps necessary to make the control framework feasible in practice. Furthermore, we will present more thorough comparisons of our work to procedural methods for adding high-frequency details such as the wavelet turbulence method [58]. We will also further explore the artistic effects and visual improvements achievable with various, time-dependent guiding weights. Finally, we will continue investigating the visual significance of our framework parameters such as the lowpass filter size.

Read papers II and III at this point.

7.4 Further Details on Customized Multigrid Solver

Recall that the multigrid method [14] solves a linear system $A\vec{X} = \vec{B}$ on progressively coarser grids. It transfers solutions from coarser to finer grids and residuals from finer to coarser grids by means of interpolation and restriction operators, respectively. On each grid, or level, h, of the multigrid solve, a matrix operator, A^h, is constructed and a relaxation method is employed for a small number of iterations. The strength of the multigrid method comes from the fact that the low frequencies of the solution converge much faster on coarser than finer grids. Therefore it makes sense to transfer the residual of an intermediate fine solution $\vec{V}$ to a coarser grid once the high-frequency components of the error have been eliminated. Relaxing on he coarser grid will eliminate the lower frequencies of the residual, which we can then use to correct $\vec{V}$. This idea is formalized in the numerical scheme called a *V-Cycle* which is heavily utilized by the multigrid method. A V-Cycle on level h is thus comprised of the following recursive pseudocode, where R_h^{2h} and I_{2h}^h are restriction and interpolation operators, respectively:

$\texttt{VCycle}^{\texttt{h}}(\vec{V}^h, \vec{B}^h)$:

1. Relax a number of times on $A^h\vec{X}^h = \vec{B}^h$ with the given initial guess $\vec{V}^h$.

2. If h is the coarsest grid, go to step 4.
   Otherwise, set
   $$\vec{B}^{2h} \leftarrow R_h^{2h}(\vec{B}^h - A^h\vec{V}^h),$$
   $$\vec{V}^{2h} \leftarrow \vec{0},$$
   $$\vec{V}^{2h} \leftarrow \texttt{VCycle}^{\texttt{2h}}(\vec{V}^{2h}, \vec{B}^{2h}).$$

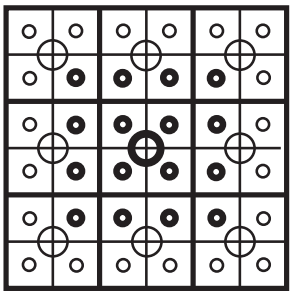
3. Correct $\vec{V}^h \leftarrow \vec{V}^h + I_{2h}^h\vec{V}^{2h}$.

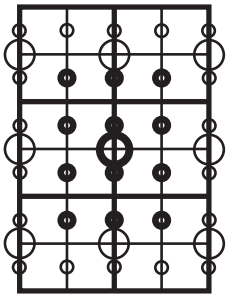
4. Relax a number of times on $A^h\vec{X}^h = \vec{B}^h$ using the corrected $\vec{V}^h$ as initial guess.

For our linear system — shown in figure 3 of paper II — we found that extending the solver with restriction and interpolation operators that operate on cell faces as well as cell centers (in order to maintain a staggered grid on each level of the multigrid solve) improved the ratio of errors in each iteration roughly by a factor of two and required only a few multigrid cycles to converge. However, to do this properly, new interpolation and restriction operators are required. We first describe how to construct these operators, and then we describe how to couple them to our linear system.

7.4.1 Interpolation and Restriction Operators

As explained above, better multigrid convergence can be obtained if the staggered grid arrangement is maintained at all levels, $\Delta x = h, 2h, 4h, \ldots$, of the multigrid solve. This requires specialized interpolation and restriction operators. We have designed new operators from the variational property that the interpolation and restriction operators are





$$(I_\lambda)^{(i,j)} = \frac{1}{16}\begin{pmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{pmatrix}, (R_\lambda)_{(i,j)} = \frac{1}{64}\begin{pmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{pmatrix} \quad (I_x)^{(i,j)} = \frac{1}{8}\begin{pmatrix} 1 & 2 & 1 \\ 3 & 6 & 3 \\ 3 & 6 & 3 \\ 1 & 2 & 1 \end{pmatrix}, (R_x)_{(i,j)} = \frac{1}{32}\begin{pmatrix} 1 & 2 & 1 \\ 3 & 6 & 3 \\ 3 & 6 & 3 \\ 1 & 2 & 1 \end{pmatrix}$$

Figure 7.2 **Top left:** A coarse 3×3 grid superposed on top of a finer 6×6 grid. The cell-centered grid points in the coarse grid are shown with large circles and the cell-centered grid points in the finer grid are shown with small circles. The single bold grid point in the coarse grid contributes to all bold grid points in the finer grid when interpolating from the coarse to the fine grid. **Bottom left:** Shows the corresponding interpolation- and restriction-weights corresponding to the nonzero entries in the (i,j)'th column of I_λ and the (i,j)'th row of R_λ. **Top right:** A coarse 2×3 grid superposed on top of a finer 4×6 grid. The single bold grid point in the coarse grid contributes to all bold grid points in the finer grid when interpolating from the coarse to the fine grid. **Bottom right:** Shows the corresponding interpolation- and restriction-weights.

transposes of each other, meaning that $I_{2h}^h = c(R_h^{2h})^T$ [14], where I_{2h}^h is the interpolation operator that transfers a function on level $2h$ to level h and R_h^{2h} is the restriction operator that transfers a function from level h to level $2h$. Essentially this property implies that if a grid point at level h affects a grid point at level $2h$ through the restriction operator, then the grid point at level $2h$ will also affect the grid point at level h through the interpolation operator. When interpolation and restriction operators satisfy this variational property they are optimal with respect to a two-grid correction process in the sense that the error at the fine grid after correction is minimal among operators of the given order [14].

Next we will show how the filters are derived in two dimensions. During a multigrid solve for a two-dimensional simulation, three grids are active: The grid of x-velocities, $\mathbf{G}_x$, living on faces with normals in the x-direction, the grid of y-velocities, $\mathbf{G}_y$, living on faces with normals in the y-direction and the grid of cell-centered Lagrange multipliers $\mathbf{G}_\lambda$. For each of these grids we define a unique interpolation and restriction operator. The operators are defined with subscripts matching those of the grids, such that *e.g.* R_λ and I_λ are the restriction and interpolation operators of $\mathbf{G}_\lambda$, respectively. To define a pair of interpolation and restriction operators, we start by defining the interpolation operator and then use the aforementioned variational property to determine the restriction operator. The interpolation operator is simply defined by a bilinear interpolation between a coarse and a fine grid as shown in figure 7.2, where a coarse grid is superposed on top of a twice as fine grid. In particular the interpolation operator, I_λ, must interpolate cell-centered entities (left part of figure 7.2) from cell centers in the coarse grid to cell centers in the fine grid, and I_x must interpo-

late face-centered entities (right part of figure 7.2) from face centers in the coarse grid to face centers in the fine grid. I_y is derived similarly, and the corresponding restriction operators are defined from the variational property described above with a scale, c, ensuring that the weights sum to one. Finally, to extend to three dimensions, a trilinear interpolation is simply used instead of a bilinear interpolation.

7.4.2 Coupling the Operators to the Linear System

Given the matrix operator, A^h, at level $\Delta x = h$ of the multigrid solve, we need to compute the matrix operator, A^{2h}, at the next coarser level $\Delta x = 2h$. This is described in detail for constant interpolation and restriction operators in [14], and in particular $A^{2h} = R_h^{2h} A^h I_{2h}^h$, where R is the restriction operator and I is the interpolation operator. However, in our case, the restriction and interpolation filters vary for the face-centered and cell-centered grids. Next, we explain how A^{2h} is computed in our case. For this purpose it is convenient to first rewrite the matrix in figure 3 of paper II in a blocked matrix format[2]. Let $\mathbf{a} = \mathbf{v}(\mathbf{r}) + \frac{(1-\alpha)}{\alpha} \int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r}) [\mathscr{F} * \mathbf{v}](\mathbf{q}) d\mathbf{q}$, $\mathbf{b} = \frac{(1-\alpha)}{\alpha} \int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r}) [\mathscr{F} * \mathbf{v}](\mathbf{q}) d\mathbf{q}$, $\mathbf{c} = \tilde{\mathbf{v}} + \frac{(1-\alpha)}{\alpha} \int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r}) \mathbf{v}_{\text{low}}(\mathbf{q}) d\mathbf{q}$, $\mathbf{d} = \frac{1}{\alpha} \nabla\lambda(\mathbf{r})$ and $\mathbf{e} = \frac{1}{\alpha} \Delta\lambda(\mathbf{r})$. Next, let $\mathbf{v}_n$ denote the nth component of vector $\mathbf{v}$. Then the matrix can be represented in the following blocked matrix format for a two-dimensional simulation (the situation is similar for three dimensions):

$$\left(\begin{array}{c|c|c} \mathbf{a}_1 & \mathbf{0} & \mathbf{d}_1 \\ \hline \mathbf{0} & \mathbf{a}_2 & \mathbf{d}_2 \\ \hline \nabla\cdot\mathbf{b}_1 & \nabla\cdot\mathbf{b}_2 & \mathbf{e} \end{array}\right) \left(\begin{array}{c} \mathbf{v}_1 \\ \hline \mathbf{v}_2 \\ \hline \lambda \end{array}\right) = \left(\begin{array}{c} \mathbf{c}_1 \\ \hline \mathbf{c}_2 \\ \hline \nabla\cdot\mathbf{c} \end{array}\right)$$

Each block-operator in this matrix operates on values in one grid, $\mathbf{A}$, *e.g.* a cell-centered grid, and transfers them to values in another grid, $\mathbf{B}$, *e.g.* a face-centered grid, which we illustrate by $\mathbf{A} \rightarrow \mathbf{B}$. The specific relationships are illustrated in the following blocked marix:

$$\left(\begin{array}{c|c|c} \mathbf{G}_x \rightarrow \mathbf{G}_x & \mathbf{G}_y \rightarrow \mathbf{G}_x & \mathbf{G}_\lambda \rightarrow \mathbf{G}_x \\ \hline \mathbf{G}_x \rightarrow \mathbf{G}_y & \mathbf{G}_y \rightarrow \mathbf{G}_y & \mathbf{G}_\lambda \rightarrow \mathbf{G}_y \\ \hline \mathbf{G}_x \rightarrow \mathbf{G}_\lambda & \mathbf{G}_y \rightarrow \mathbf{G}_\lambda & \mathbf{G}_\lambda \rightarrow \mathbf{G}_\lambda \end{array}\right)$$

For the blocks that transform between different grids, different types of interpolation and restriction operators must be used. In particular the grid on the left side of the arrow denotes which interpolation operator to choose, and the grid on the right side of the arrow denotes which restriction operator to choose. Hence the operator matrix A^{2h} is defined as follows:

$$\left(\begin{array}{c|c|c} R_x \mathbf{a}_1 I_x & \mathbf{0} & R_x \mathbf{d}_1 I_\lambda \\ \hline \mathbf{0} & R_y \mathbf{a}_2 I_y & R_y \mathbf{d}_2 I_\lambda \\ \hline R_\lambda \nabla\cdot\mathbf{b}_1 I_x & R_\lambda \nabla\cdot\mathbf{b}_2 I_y & R_\lambda \mathbf{e} I_\lambda \end{array}\right)$$

where the restriction and interpolation operators are as defined in the previous section. The coupling to the linear system of paper III is performed in a similar manner.

[2]Note that we in the following will use a notation consistent with paper II and hence deviate a little from the normal notation of this part of the dissertation.

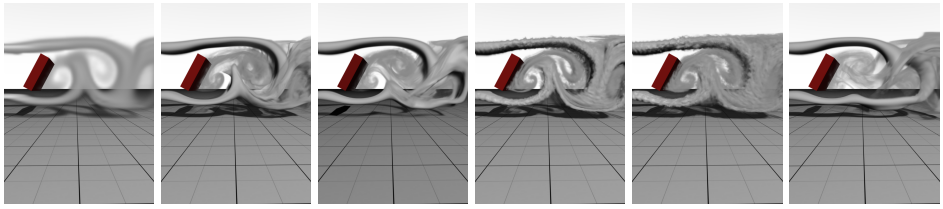


Figure 7.3 Using our guiding method, stable laminar flow is preserved and instabilities arise near boundaries at high resolution whilst maintaining correspondence to the low resolution simulation. Images from left to right: Low resolution, our method guiding with eroded densities, our method guiding with eroded densities combined with the error estimate, the wavelet turbulence method with strengths 0.2 and 0.5, and finally the high resolution simulation.

7.5 Discussion and Evaluation

We end the chapter with a discussion of some issues that were not fully covered in papers II and III. Parts of these considerations will be included in our journal paper on fluid control, which is in preparation.

While our framework generally ensures correspondence between the velocity fields of the low-resolution prototype simulation and the high-resolution final simulation, it can still be hard to create and model the solution that we want as the low-resolution, input simulation (assuming that it is even possible to achieve the desired motion in low-resolution). For instance, it took a lot of effort to model the problems in figure 7 of paper II to get a satisfactory result. Setting up fluid simulations requires experience and talent, and it would be a thrilling prospect to evaluate our method with professional artists.

In some situations, using the error estimate or the eroded densities as guiding weights can ruin the correspondence between the velocity fields. Generally, our method does not guarantee correspondence of smoke densities. This can be an issue since numerical diffusion of the smoke densities is very high at low resolutions, meaning that the resemblance between the guiding and the guided simulation can become lost. It would be interesting to add a term to the minimization problem for handling this, similar to the smoke gathering term of Fattal and Lischinski [25]. Alternatively, we could combine our method with their approach using an upsampled, filtered version of the low-resolution smoke density field as the target shape for each iteration.

In our upcoming journal paper, we have performed more thorough comparisons of our method to simpler and faster alternatives such as the wavelet turbulence method of Kim *et al.* [58], and the method of explicitly blending guiding velocities into the current velocity field before presure projection[3]. Figure 7.3 compares our method to the wavelet turbulence method. It illustrates an important difference between our method and many procedurally generated flows: Our method preserves the stable, laminar flow to the left and immediate right of the solid boundary, while the wavelet turbulence method forces it to become turbulent. Figure 7.4 shows the visual difference

[3] Thürey *et al.* [123] suggested a similar approach for SPH and LBM where Lagrangian particles were used to define the guiding velocities.

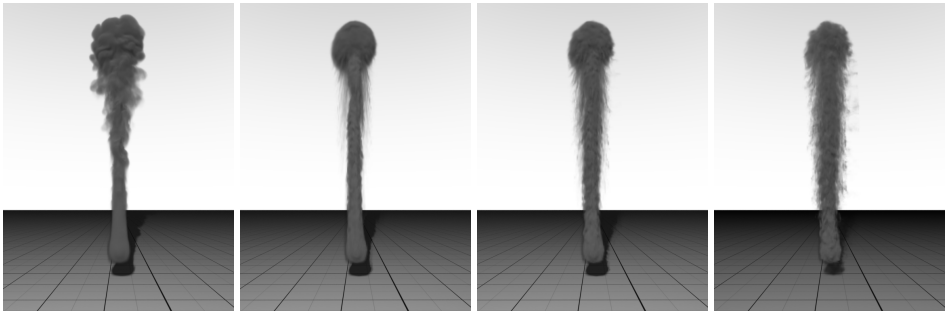


Figure 7.4 Images from left to right: Our method guiding with eroded desities and the error estimate, the wavelet turbulence method with strengths 0.2, 0.5 and 1.0, respectively.

between our method and wavelet turbulence. The post-processing nature of the wavelet turbulence method makes it incapable of producing larger-scale features not present in the low-resolution input, such as the turbulent vortices on the left and right sides of the smoke column in the leftmost image.

In paper II we compared our approach to the faster and simpler method of explicitly blending the low frequencies of the high-resolution velocity field with the guiding velocity field. Figure 7.5 shows a further exploration of this alternative approach. In this more elaborate comparison we see that the visual artifacts, originally identified in our paper, seem to persist even as the blend factor is increased to diminish the influence of the guiding velocity field. However, at blend factor 0.85, the artifacts do not show up until quite late in the simulation, which makes this approach very attractive for simulations that are not too long. The same is true for blend factor 0.90. Blend factor 0.95 does not seem to result in artifacts at all, but here the guided high-resolution simulation begins to lose resemblance to the guiding low-resolution simulation.

Figure 7.6 explores the effect of changing the lowpass guiding filter. We see that more turbulence develops as the size of the filter is increased. Visually, the simulations with larger filters also begin to lose correspondence to the original, low-resolution simulation due to the high amount of turbulence. The timing seems to remain intact though.

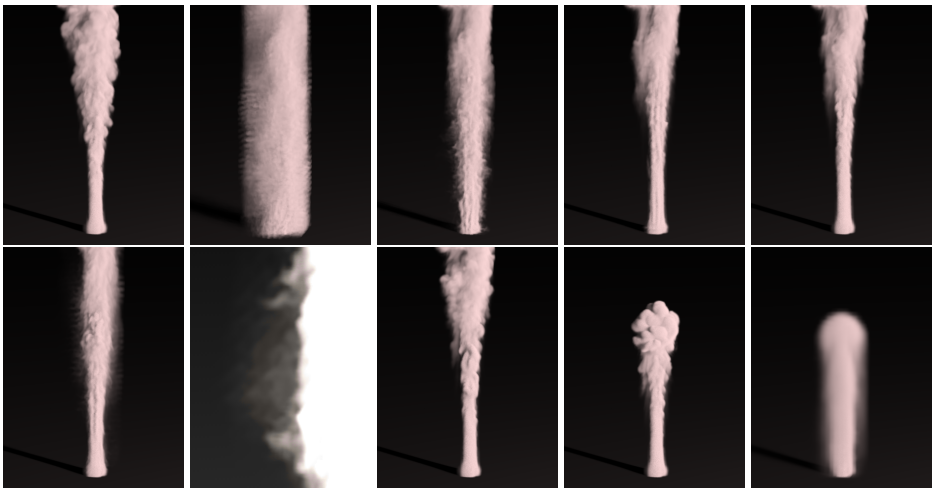


Figure 7.5 **Top:** Frame 399 from the following simulations: Our guiding method with eroded densities combined with the error estimate. The next four simulations are performed by explicitly blending the velocity field with the guiding velocity field before pressure projection with blend factors 0.25, 0.65, 0.75 and 0.85, respectively. **Bottom:** Images from left to right: Frame 1599 from explicitly blending the velocity field with the guiding velocity field using blend factor 0.85. At frame 399 no artifacts had appeared (rightmost image in top row), but at frame 1599 the artifacts start to appear again. The next image shows a close-up of frame 4999 with blend factor 0.90. The intensity has been increased to more clearly highlight the artifacts appearing with this blend factor. The following image shows frame 399 using blend factor 0.95. In this case no artifacts have appeared, but as shown in the next image (frame 220, blend factor 0.95), the column also starts to rise faster than the low resolution simulation shown in the last image. A blend factor of 0.95 does not seem to generate artifacts, at least for the first 5000 iterations that we ran the simulation.

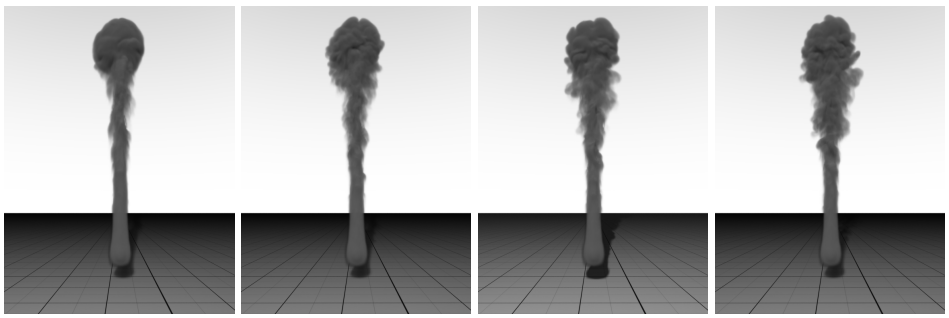


Figure 7.6 Images from left to right: Our method guiding with eroded densities and the error estimate with lowpass guiding filters of dimensions 5^3, 7^3, 9^3 and 11^3, respectively.

CHAPTER 8

CONCLUSION

This chapter concludes the overview part of this dissertation. We have explored two distinct approaches to improving the management of high-resolution simulations within scientific computing and computer graphics. In particular, we have proposed an out-of-core framework for efficient streaming of level set computations. Our contributions included the development of code transformations that better utilize the memory hierarchy in order to sustain performance when using external memory. Furthermore, we have proposed a guiding framework for controlling fluid simulations. We derived new mathematical models for coupling simulations of differing resolution in order to allow for a more efficient workflow. With it, the visual artist can design the desired flow at low resolution and then reliably achieve the same overall flow at high resolution with added high-frequency detail.

The next section covers ideas for future work, while section 8.2 rounds off by revisiting the research objectives stated in chapter 1 and summarizing our contributions.

8.1 FUTURE WORK

While the work presented in this dissertation has demonstrated efficient level set computations at very high resolution, the demand for higher resolution simulations at lower simulation times is ubiquitous. Similarly, our fluid control framework and other control paradigms have only scratched the surface with regard to what can be done to improve the workflow surrounding fluid simulations for visual effects. Further research should be conducted to reduce the causes of frustration for artists as well as the expenses in terms of time and money for film productions. Below we outline directions for future research which we believe are likely to contribute to the areas of level sets and control of fluid simulations in the future.

On the Out-of-Core Framework

Automatic, Dynamic Tiling and Concatenation of Computations: Determining and adjusting the tile sizes and tiling directions automatically depending on the available amount of main memory would be highly useful. It would ensure that the entire memory is utilized at all times, but more importantly, it could be used to ensure load balancing of the CPUs when simulating several tiles in parallel. As it is now, we assume that it is possible to provide an initial configuration that remains valid throughout the simulation. In a similar vein, an automatic determination of the number of computations that can be concatenated within memory, would ensure that no excess disk bandwidth is used. It would possibly require an initial pass through data combined with a statistical model of the upcoming memory requirements in order to perform these dynamic adjustments.

Further Reduction of Computation Time: Our framework remains CPU limited even for very high resolutions, and thus performs nearly as well as an in-core simulation. However, the sheer number of grid points computed on means that computation times themselves are now the limiting factor. We believe that the small overhead introduced by our framework can be reduced further through careful engineering of the iteration stencils. Additionally, we wish to reduce computation time by further investigating parallelization over multiple CPU cores. This could be combined with implicit methods for solving the PDEs in order to allow for larger time step sizes, hence reducing the number of computations required to integrate the solution to a certain point in time.

Combination With Fluid Simulation: It would be interesting to explore strategies similar to the skewing and tiling transformations employed by our out-of-core framework in the context of fluid simulations. We propose to combine the simple, local steps of the LBM [15] with the DT-Grid as the underlying structure for the grid. Since the LBM does not depend on global information, it should be possible to stream a fluid simulation step. Also, because no global information is needed to compute the velocity field for the next level set advection step, we should be able to perform several combined fluid and level set simulation steps during one pass (if we are able to conservatively estimate the maximum velocity or use an unconditionally stable advection scheme). Of course, this combination of fluid and level set simulation only applies if we want to use a level set as the surface representation. The LBM can easily be extended to handle a free surface by introducing a layer of special interface cells, and since the framework allows for high resolutions, we can get a finely resolved surface this way instead. The streaming approach of our method would thus still be applicable.

On Fluid Control

Free Surface Fluid Control: Our fluid control framework exclusively deals with smoke animations, *i.e.* fluid simulations where there is no dynamically evolving free

surface which changes the simulation domain. Of course, it would be interesting to extend the framework to work with free surface fluids. The major challenge lies in the fact that the free surface of the low-resolution simulation will differ from the free surface of the high-resolution simulation due to numerical dissipation. We propose to handle this discrepancy by simply disabling guiding of the high-resolution velocity field outside the regions corresponding to the low-resolution liquid. Alternatively, we could extrapolate the velocities of the low-resolution simulation across the low-resolution free surface and then smoothly diminish guiding outside the regions corresponding to low-resolution liquid.

Density Matching: Our fluid control framework only ensures correspondence between the *velocity fields* of two simulations at different resolution. Hence, our method does not guarantee correspondence of the smoke densities. Due to the high numerical diffusion of smoke densities at low resolutions, the resemblance between the guiding and the guided simulation can be lost. In the future, we would like to add a term to the minimization problem for handling this, similar to the smoke gathering term of Fattal and Lischinski [25]. Alternatively, we could experiment with combining our method with their approach using an upsampled, filtered version of the low-resolution smoke density field as the target shape for each iteration of the high-resolution simulation.

Parameters and Exploration: Since one of the problems artists face when working with fluid simulations is the non-intuitive meaning of the parameters, it seems that providing more intuitive parameters would yield better control for the artists. One example of such an intuitive parameter could be a measure of how turbulent the flow should be. The parameter could be controlled by a simple slider which in turn controls how to interpolate pre-recorded data perhaps based on a frequency analysis or dimensionality reduction. Also, an ability to adjust parameters on the fly in the middle of a simulation seems like a natural way to expose control. This could be coupled with the ability to explore different possible futures of the simulation given small variations in the parameters akin to the many-worlds browsing approach of Twigg and James [129].

Painting the Control: Another way to let the artists control the course or modify the result of a simulation would be to allow them to "paint" various properties onto the liquid surfaces. This would in turn require a framework for propagating the "paint" (*e.g.* a function) defined on the surface as the surface itself propagates. The approach could be combined with the idea of controlling the fluid surface movement through modification of the velocity extrapolation proposed by Enright *et al.* [24]. If the approach was generalized to enable drawing everywhere in the embedding domain, it could be used to draw paths for control particles similar to the ones used by Thürey *et al.* [123]. Similarly, it could be used to generate guiding velocity fields for our proposed fluid control framework. An artist could modify an existing simulation by painting curves in order to enhance or reduce a certain motion.

Practical Evaluations: We believe that a lot of insight could be gained by performing practical evaluations of our fluid control framework using artists from the visual effects industry. Within computer graphics in academia there is no strong tradition for doing this, and we foresee the need for multidisciplinary research collaborations in order to do this properly.

The above list of directions for future work is in no way exhaustive. It does however present a number of outstanding issues that lie in direct continuation of our work up to now. In several of the examples, we have already begun preliminary investigations.

8.2 Final Thoughts

Because of their many advantages, including the capacity to handle arbitrary topological changes, level sets have become prevalent within computer graphics, scientific computing and visual effects production. Therefore, existing level set schemes are continually being challenged and pushed to the limits as the demand for higher resolutions and more details becomes universal. Within fluid simulation for computer graphics, the request for visually rich and complex effects has led to frustrating and expensive workflows. This is primarily due to these simulations being computationally intensive and notoriously hard to control. These facts prompted us to pursue the two research objectives stated in the introduction. We will revisit them here and evaluate our success in achieving them. The objectives are as follows:

- To improve the feasibility of using level sets at high resolutions in computer graphics and scientific computing applications.
- To improve artistic control and efficiency in the typical workflow surrounding fluid simulations in visual effects.

In paper I we propose an efficient out-of-core framework for level set simulations. By code transforming the algorithms using skewing and tiling we reduce the number of passes over the data during computations. Thus we only need one pass over the data for each level set computation, which comprises a deformation step, a reinitialization step and a rebuild of the narrow band. This results in a significant improvement of the previous framework [91], on which we have based our work. The new framework is CPU limited and runs at speeds of up to 92% of an in-core simulation. It is independent of disk latency and can handle resolutions as high as allowed by the available disk space. Furthermore, it facilitates parallel computation on the individual tiles. The objective of improving the feasibility of using level sets at high resolutions in computer graphics and scientific computing applications has, however, not been met entirely as these high resolutions often require too much computation time to be useful for computer graphics and visual effects. However, we have demonstrated that high-resolution simulations can be run even on very limited hardware, thus reducing the cost of performing large computations, which is another aspect of making it feasible in practice. While we have focused exclusively on level set computations, our developed techniques are relevant for general out-of-core stencil based computations.

In papers II and III we present a fluid control framework that couples fluid simulations at different resolutions. This allows artists to efficiently prototype simulations at low resolution and then — ideally — only run the costly high-resolution simulation once in order to get the final result. However, it also allows an artist to concentrate on exploring the simulation parameters in high resolution without worrying about changing the approved bulk motion from the low-resolution starting point. Our mathematical model minimizes the difference between the low frequencies of the high-resolution velocity field and the guiding low-resolution velocity field, thus ensuring correspondence in most cases. We propose several methods for determining the guiding weights, which dictate the strength of the coupling, and we demonstrate several artistic effects accomplished using the framework. Still, the achievement of the second objective is hard to evaluate without performing practical evaluations of our framework.

We hope that the research proposed in this dissertation will find usage both within the research community around level sets and fluid simulations, and also in the visual effects industry. The fluid control framework in particular, was developed with a major effects company and will hopefully see practical usage there in the future. The algorithms and mathematical models presented are likely to form an integral part of much of our future research, and we foresee many exciting extensions and improvements in the near future.

PART II

PAPERS

PAPER I

OUT-OF-CORE COMPUTATIONS OF HIGH-RESOLUTION LEVEL SETS BY MEANS OF CODE TRANSFORMATION

Brian Bunch Christensen Michael Bang Nielsen Ken Museth

Abstract

We propose a storage efficient, fast and parallelizable out-of-core framework for streaming computations of high resolution level sets. The fundamental techniques are *skewing* and *tiling* transformations of *streamed* level set computations which allow for the combination of interface propagation, re-normalization and narrow-band rebuild into a single pass over the data stored on disk. This improves the overall performance when compared to previous streaming level set frameworks that require multiple passes over the data for each temporal iteration step. As a result, streaming level set computations are now CPU bound and consequently the overall performance is unaffected by disk latency and bandwidth limitations. We demonstrate this with several benchmark tests that show sustained out-of-core throughputs close to that of in-core level set simulations.

1 INTRODUCTION

While the idea of using implicit functions for interface capturing can be dated back as far as [17, 18], the level set method and the underlying numerical schemes were first proposed in [95]. Since then, it has been applied to a wide range of interface capturing problems in scientific computing and related fields. Examples hereof include the simulation of multi-phase flows [118] such as bubbles and drops, solidification

Submitted to: Journal of Scientific Computing

[37], Willmore flow [20], partial differential equations and variational problems on manifolds [8], geometric optics [97] as well as fluid animation [30] and geometric modeling in computer graphics [83].

The propagation of a time-dependent level set interface is given by partial differential equations *e.g.* of Hamilton-Jacobi type. In most cases the scalar function is sampled on a regular Eulerian grid, although recent work has also employed fully Lagrangian representations [43]. In order to adequately capture interface details and obtain sufficient numerical accuracy, a combination of high order discretization schemes and/or high resolution Eulerian grids is often required.

In cases where only a single level set is of interest (*e.g.* the zero-crossing of the interface) computations can be restricted to a narrow band of grid points surrounding the interface [1, 16, 101, 132]. More recent work combines the idea of restricting the computations to a narrow band with sparse data structures in order to reduce storage requirements and enable interfaces to be sampled on much higher resolution grids. These narrow-band data structures include blocked grids [11, 65, 81, 82], dynamic tubular grids (DT-Grid) [87, 90] and hierarchically run-length encoded grids [47]. A few authors have developed adaptive methods that do not restrict computations to a narrow band but instead refine the computational grid, typically closer to the interface [73, 76, 77, 117, 119].

Although these level set data structures enable computations on high resolution grids, they are all limited by the available main memory. Despite the fact that modern 64-bit operating systems allow for a virtual address space of 16 exabytes, RAM modules remain a relatively expensive commodity. In comparison, disk storage is two to three orders of magnitude cheaper per byte, consumes about two orders of magnitude less power per byte and offers a capacity that is typically several orders of magnitude larger [66]. Hence algorithms capable of utilizing disk space, often referred to as *out-of-core* or *external memory* algorithms, have the potential of higher resolution simulations at lower costs. However, disk storage has much higher latency (referencing a single data item is four to five orders of magnitude slower than main memory access), and the development and study of efficient external memory algorithms has emerged into a field of its own [125, 130]. Note that the ongoing development of solid state drive technology offers promising speed improvements for random access to external memory in the future.

Recently Nielsen *et al.* [91] proposed an out-of-core framework for narrow band level set simulations based on the DT-Grid data structure, and this paper significantly improves on that work. The main contributions of the out-of-core framework presented in [91] are prefetching and page-replacement algorithms designed for stencil based level set computations. Whilst allowing for grid resolutions only limited by the available disk storage, that method remains IO limited, and the throughput (measured in computed grid points per second) drops to 42% of in-core simulation throughput for some numerical schemes. One of the main reasons for the IO limitation in [91] is the fact that each step in the level set simulation requires the data to be streamed to and from disk multiple times. A typical time-step actually requires between 5 to 10 passes over the data. In contrast, the method proposed in this paper requires data to be streamed only once per time-step. In fact, for simulations with certain properties,

data is only required to be streamed once for a number of subsequent timesteps, hence reducing bandwidth usage further. As a result, our new out-of-core algorithms are CPU limited as opposed to IO limited and exhibit a sustained, *i.e.* resolution independent, throughput of $77-92\%$ (depending on the numerical scheme) of the throughput obtainable by internal memory simulations[1].

Our general approach is to leverage on established theory from the area of compiler algorithms which performs *code transformations* that optimize cache *locality*, *i.e.* minimize the number of times a given data element is loaded into the cache from main memory. In particular, we employ the mathematical model of reuse and locality developed by Wolf and Lam [134]. Applying code transformations to out-of-core as opposed to in-core level set simulations poses unique challenges since our goal is to minimize the number of times a given data element is transferred from disk to main memory. The ratio of disk to main memory latency is much higher than the corresponding ratio of main memory to cache latency. Consequently a data layout that works well in-core may need to be redesigned for out-of-core application, although they may have the same asymptotical IO complexity.

Our contributions can be summarized as follows. In this paper we prove and demonstrate by implementation that the finite difference (FD) schemes used for level set simulations, HJ ENO [42], HJ WENO [50, 52, 70], BFECC [21] and TVD RK [112], have data reuse both temporally and spatially. However, locality is not directly implied. To improve locality we derive code transformations based on skewing (*i.e.* shearing the iteration space by a linear transform) and tiling (*i.e.* partitioning) and prove that these transformations maximize locality both spatially and temporally in the model of Wolf and Lam [134]. In particular, tiling alone is not sufficient to optimize locality for the FD stencil based level set simulations. Code transformations are applied to all steps in the narrow band level set computation, including propagation/advection, re-distancing and narrow band rebuild. In this way only a single pass over the data is required for each time-step or sequence of time-steps. Additionally we propose a tiled version of the Fast Iterative Method [49] which enables fast out-of-core solution of the Eikonal re-distancing equation $|\nabla\phi| = 1$ for narrow band level sets. To reduce memory requirements during simulation, we propose an in-core storage mapping for the intermediate values associated with the skewing transformation that is linear in the number of intermediate values. Furthermore, we also propose a linear out-of-core storage mapping associated with the tiling transformations that partitions the narrow band into tiles and boundary grids and facilitates computation on each tile independently and in parallel. Our framework is optimal with respect to streaming to and from disk in both the IO model [2] and the Cache Oblivious model [33], as only sequential stencil access is required. The sequential access enables utilization of the cache coherent DT-Grid in combination with the page-replacement and prefecthing algorithms from [91]. Finally, the resolution of the computational grids utilized in our method is limited only by the amount of disk space available.

To illustrate the feasibility of our new out-of-core framework we document its

[1] Our out-of-core technique introduces a minor computational overhead compared to regular in-core simulations, which explains why the efficiency is still below 100% even though the method is not IO limited.

performance with benchmarks of several different types of level set simulations and numerical schemes. Additionally, we have used our framework for several applications of the level set method on high resolution computational grids. This includes advection in a divergence free velocity field and surface smoothing by mean-curvature flow of a model with an effective grid resolution of $17149 \times 9987 \times 5734$. In order to further improve simulation times, we also demonstrate a multi-core implementation of our out-of-core framework.

The remainder of this article is organized as follows: Section 2 first summarizes related work. In section 3 we provide an overview of our out-of-core framework and its individual components, and introduce the terminology. Next we describe our proposed skewing and tiling transformations and the associated storage mapping schemes. Detailed descriptions of the transformations along with proofs of their locality properties are provided in appendix A. Section 4 then proceeds to present benchmark results. Finally, section 5 demonstrates some applications of our framework and section 6 concludes and outlines directions for future work.

2 Related Work

In this section we review related work in the areas of out-of-core algorithms, simulation and loop transformation theory and algorithms. Out-of-core, or streaming, algorithms are applicable in all areas of computational science and scientific computing that involve massive data sets not feasible for storage in main memory due to hardware and cost limitations. These application domains include image repositories, digital libraries, relational and spatial databases, computational geometry, simulation, linear algebra and computer graphics. For a general survey of external memory algorithms see [130], and for a specific survey in the area of linear algebra and simulation we refer to [125].

Despite its potential for large scale simulation, we are only aware of a surprisingly small body of previous work directly related to out-of-core simulations. Pioneering work was done by Salmon and Warren [106] for N-body simulation in astrophysics. Their work was based on tree data structures and applied reordered traversals and a Least-Recently-Used page-replacement policy for efficiency. Bibireata *et al.* [9] use loop fusion and tiling (see below) to perform out-of-core tensor contractions for simulations of electron structures. Trac and Pen [48] proposed an out-of-core algorithm for Eulerian grid based cosmological simulation. In their method, global information is computed on a low resolution grid that fits entirely in memory, whereas local information is computed on an out-of-core high resolution grid tiled into individual blocks that fit into memory. The individual blocks are loaded and simulated in parallel for a number of time steps and then written back to disk. More recently Nielsen *et al.* [91] proposed a combined framework for compression and out-of-core simulation of Eulerian grid based level sets and fluids based on the DT-Grid data structure [90]. A fundamental property of their work is that existing level set and fluid simulation code does not have to be re-written, and hence focus is on developing prediction schemes for statistically based compression as well as prefetching and replacement strategies

for stencil based access. However, a consequence of this property is that the method remains IO limited, as the Eulerian grids need to be streamed through memory several times for each step of the simulation.

In the field of compiler algorithms and cache coherency a lot of attention has been devoted to the so-called "loop transformation theory" - see [62] for a comprehensive overview. In particular Wolf and Lam [134] propose a theory of reuse and locality as well as an automatic algorithm for improving the data locality of loop nests by applying a sequence of loop transformations. The loop nest is analyzed for reuse and the resulting loop transformation is found as the maximum of an objective function measuring data locality of equivalence classes of localized iteration spaces. This theory formed the basis for the optimized loop transformations suggested by [74].

Wonnacott, [135], introduced a particular type of transformation denoted time skewing and an associated storage scheme that takes advantage of the available cache memory. The transformation results in locality both in the spatial and temporal dimensions of the iteration space and is proposed for in-core time step stencil computations. Computations are active only on a wavefront of grid points – the *wavefront of execution* – that can be fitted into the cache via the proposed storage scheme. Since this wavefront is skewed with respect to time in all spatial dimensions it is not well suited for implementation on a sparse narrow band data structure such as the DT-Grid. This is due to the fact that the skewed wavefront iteration order does not correspond to sequential access into the underlying data structure. In contrast the loop transformations we propose in this article can be implemented as sequential access which is faster than random access for this type of data structure. Time skewing along with various other optimization approaches for stencil computations, both cache aware and cache oblivious, were also studied by Kamil *et al.* [53].

Kandemir *et al.* [54] present a unified optimization framework that targets perfectly nested loops of computations running out-of-core and in parallel. In particular their framework is intended for integration in a compiler and it optimizes for locality of data references, array file layout, parallelism and reduction of communication overhead simultaneously. Their method considers only tiling for improving data locality and constructing file-layouts, which is too restrictive in order to obtain temporal locality for stencil-based level set computations. More recently, Kandemir *et al.* proposed an I/O-Conscious Tiling Strategy for Disk-Resident Data Sets [55]. Their method focuses on adapting traditional tiling algorithms for scientitic computation loop nests to out-of-core computations in order to obtain higher IO performance. In particular, they show that both loop and data transformations are often required to achieve this goal. Again only tiling transformations are considered, and the class of algorithms investigated does not include stencil-based computations.

Song and Li [113] present a scheme which optimizes cache locality for a certain class of nontrivial imperfectly-nested loops. They propose a number of loop transformations to enable tiling. Specifically, they provide a compiler algorithm for optimizing skewing of the spatial dimensions subject to dependencies, automatically selecting the optimal tile size, and deducing an efficient storage scheme through array duplication. Their method is, however, limited to computations which only depend on references to values from the same or the previous iteration of the time step loop. When using

higher order temporal schemes such as TVD RK, our computations do not adhere to this limitation.

3 Skewing and Tiling Level Set Computations and Data Structures

Figure 1 provides an overview of our out-of-core level set framework, where the components corresponding to our contributions are highlighted in blue. Central to this framework is a tiled DT-Grid data structure shown in the middle of figure 1 and in figure 2. This data structure partitions a narrow band level set into a number of axis aligned tiles, storing only grid points that are part of the narrow band inside each tile (shown in yellow). Boundary grids (shown in red) are extracted from the narrow band on the boundary of each tile and used by the skewed level set computations on adjacent tiles. This effectively reduces bandwidth requirements since level set computations on a single tile only access the tile itself plus a number of small boundary grids, as opposed to streaming all the adjacent tiles. The size of the boundary grids is a function of the size of the stencil used in the FD computations as well as the number of level set steps performed on each tile. Generally the width of a boundary grid is small compared to the size of a tile. Hence, the overhead associated with the boundary grids is a small fraction of the total storage requirements and computation time. As we show in appendix A, given an N-dimensional grid, it is only necessary to tile in $N-1$ directions in order to maximize locality in both the temporal and spatial dimensions. Hence we always leave the x-direction untiled as shown in figure 2. Generally we tile in as few spatial dimensions as possible and at most $N-1$ dimensions as noted above. For example, if the required $(N-1)$-dimensional slices of the N-dimensional grid fit in memory, we do not tile the grid. The level set surfaces in each tile and boundary are stored separately as narrow bands in DT-Grid data structures using the out-of-core framework introduced by Nielsen *et al.* [91]. This is illustrated by the separated tile and boundaries beneath the rightmost arrows in the center of figure 1. The topology and values of each DT-Grid are stored independently as indicated by the layered boxes in the rightmost part of figure 1. Storing a particular component (values or topology) is managed by a *Storage Handler* which streams data either to memory or disk. In the case of streaming to disk, pre-fetching and page-replacement algorithms designed for stencil-based access are implemented by a *Storage Cache* [91]. Note that the level sets stored in tiles and boundaries may not represent closed surfaces. As long as the level set surface is intersected by convex tiles (*e.g.* axis aligned boxes), the DT-Grid data structure supports this [89].

Skewed level set computations can be performed on each tile independently and hence multiple computational threads can process separate tiles in parallel as indicated by the layered boxes in the leftmost part of figure 1. All computations take place on a single and partially in-core DT-Grid data structure storing only the active $(N-1)$-dimensional slices. This ensures that the in-core level set computations are cache efficient [89]. The in-core DT-Grid data structure is generated on the fly by merging the grid points from the tile and the boundary grids generated from adjacent tiles. This is facilitated by the lexicographic storage order of the grid points. The merging process

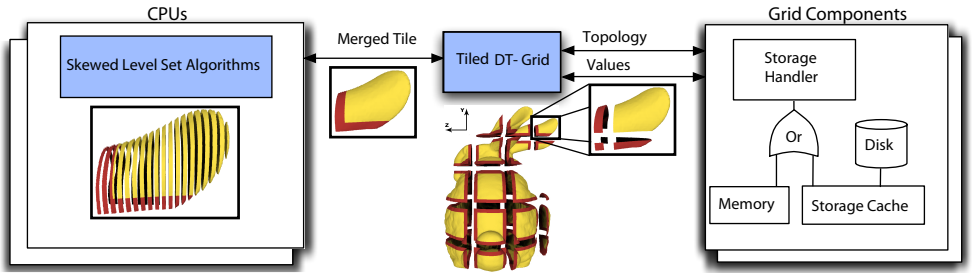


Figure 1 This figure gives an overview of our out-of-core level set framework that applies skewing and tiling transformations. **Rightmost:** Shows the components handling streaming to and from memory and/or disk, including prefetching and page-replacement [91]. **Middle:** Illustrates the central data structure, the *Tiled DT-Grid*, that implements our tiled storage mapping. The level set inside each tile (yellow) and the level sets at each tile boundary (red) are stored *separately* as narrow bands in DT-Grid data structures (middle-right) and continuously merged into a *single* narrow band during simulation (middle-left). **Leftmost:** The skewed level set algorithms process one slice of a tiled narrow band at a time.

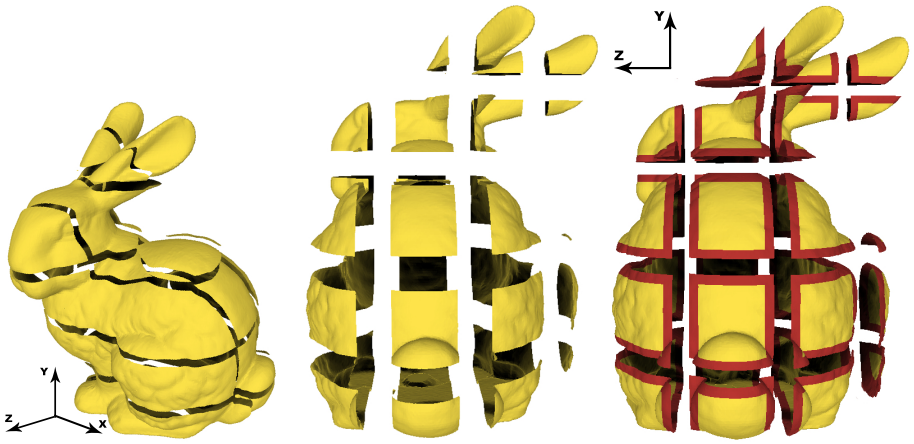


Figure 2 This figure illustrates how a level set surface of the Stanford Bunny is divided into axis aligned tiles. To maximize locality it is only necessary to tile along the y- and z-axes. The boundary grids of each tile are illustrated in red in the rightmost image.

is illustrated by the concatenated tile and boundary grids beneath the leftmost arrows in the center of figure 1. The level set computations are skewed in the spatial dimensions with respect to both the level of computation *i.e.* propagation, re-distancing and narrow band rebuild, as well as with respect to time. Hence all levels of computation, possibly at several time steps, are performed simultaneously on a tile, but in such a way that data is streamed from disk exactly once, and such that data dependencies are not violated. As indicated in the leftmost part of figure 1, computations are performed on $(N-1)$-dimensional slices of a N-dimensional tile.

The major benefit of this out-of-core approach is that, although being IO intensive, the level set algorithms become CPU bounded and hence the overall performance is unaffected by disk latency and bandwidth limitations. This means that computations

can be performed on level sets that are the size of available disks, but with a throughput close to that of in-core simulations. The reason for a small CPU overhead of our out-of-core simulations is that additional sofware layers are required to stream data to and from disk.

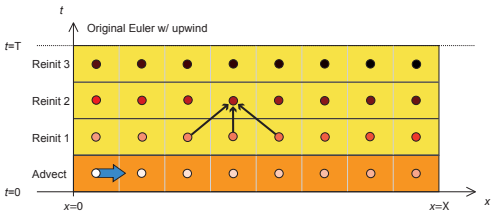
3.1 SKEWING

A time step of the overall level set iteration – a *level set step* – typically consists of an advection or propagation, a re-initialization and a narrow-band rebuild *step*, as previously mentioned. Each of these steps consists of one or several *sub-steps*, which each require one pass over the data if skewing is not applied. For example, advection with third order TVD Runge-Kutta time-integration consists of five sub-steps. In the context of code transformation theory, this situation corresponds to an imperfect loop nest since there are several loops (one for each sub-step) at the innermost nesting depth inside an outer time loop. We can convert this to a perfect loop nest by introducing a fictitious time variable and use it to distinguish which sub-step to perform in the loop body. Thus, one sub-step counts as one fictitious time step. In the leftmost side of figure 3 the front of active computations, or *wavefront of execution*, progressing through a level set step is illustrated by the orange row. The front is moving upwards, and the blue arrow indicates the progress of computations or iterations inside the front itself. To determine where the intermediate results of this execution must be stored, one employs a *storage mapping* which maps each computation on the wavefront to the address where it should store the value it produces [135]. This mapping results in a *wavefront of temporaries* which determines how much memory is required. Recall, the goal is to reduce the number of passes over the level set data (thereby minimizing data traffic) to one for a sequence of N steps. The target of this optimization lies between main memory and disk rather than the more usual CPU cache and main memory target. The motivation is that we want to eliminate the IO limitation of the previous out-of-core framework [91]. Hence, we want to perform as many steps using in-core temporaries as made possible by the amount of main memory. Therefore we need transformations of the code to make the wavefront of execution independent of the grid dimensions, such that the wavefront of temporaries fits in memory.

We perform the following analyses and transformations on full grids in one dimension for simplicity, and then generalize to N-dimensional grids in the end. In appendix A, we provide proofs of the validity of these transformations. In practice, the algorithms are implemented on sparse DT-Grids which provide constant time sequential access to neighboring grid points within a stencil.

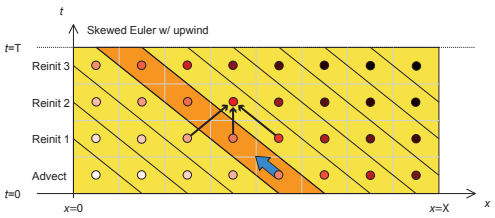
3.1.1 TRANSFORMING THE ITERATION SPACE

Typically a level set step iterates a stencil over the spatial domain of the level set while performing the relevant computations, *e.g.* solving the hyperbolic levet set equation $\frac{\partial \phi}{\partial t} - \vec{V} \cdot \nabla \phi = 0$, where ϕ is the level set function and $\vec{V}$ is a velocity field. Figure 3 shows the pseudocode of a number of sub-steps on a one-dimensional level set using simple first order upwinding in space and first order Forward Euler integration in




```
1: for t ← 0, T do
2:     for x ← 0, X do
3:         A[t+1,x] ← step(A[t,x],A[t,x−1],A[t,x+1])
4:     end for
5: end for
```

Figure 3 A number of steps on a 1D level set using the traversal outlined in the pseudocode. The orange row indicates the wavefront of execution travelling parallel to the t-axis. The black arrows indicate the dependencies of one computation.

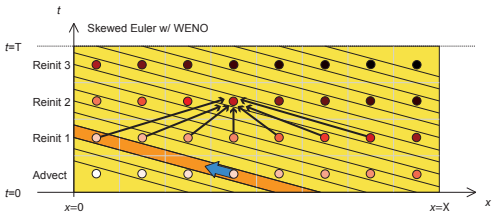



```
1: for x ← 0, X+T do
2:     for t ← 0, T do
3:         if max(t,x) < min(X+t,x+1) then
4:             w ← x−t
5:             A[t+1,w] ← step(A[t,w],A[t,w−1],A[t,w+1])
6:         end if
7:     end for
8: end for
```

Figure 4 A number of steps on a 1D level set using the transformed traversal outlined in the pseudocode. Note, that the skewed traversal order is depicted in the orginal iteration space.

time. The yellow box illustrates the iteration space of the nested loop. To simplify the following explanation, we assume that we have expanded the one-dimensional array which represents ϕ with a dimension containing the time axis, thus obtaining an array A of size $(T+1)\times X$, where T is the number of fictional time steps and X is the size of the spatial domain. The traversal order of the iteration space is indicated by the coloring of the individual iterations, starting from white over red to black. In figure 3, the entire spatial domain is traversed in each time step before moving on to the next. Not all traversals of the iteration space are valid, since a given computation $[t,x]$ has dependencies which limits the traversal possibilities. In this example, we employ a computational stencil with a width of three grid points, needed to implement the first order upwind scheme for advection and reinitialisation. As illustrated by the black arrows in figure 3 this means for example, that the result at iteration $[2,3]$ cannot be computed before the results for iterations $[1,2]$, $[1,3]$ and $[1,4]$ are known. See appendix A for a rigorous definition and analysis of dependencies. One consequence of the dependencies is that we cannot immediately interchange the t and x loops in the shown algorithm.

As mentioned, the goal is to reduce the number of passes over the level set data to one for a sequence of N steps. Therefore the algorithm cannot just iterate over the entire spatial domain for each fictional time step since the references to A in previous steps will be evicted from memory before they can be reused due to the large number of intermediate computations. We transform the code to improve this by skewing the spatial dimension of the iteration space just enough to be able to interchange the loops. Specifically, we transform the loop bounds using the transformation $T_1 : [t,x] \rightarrow [t,x+t]$ and then apply T_1^{-1} to the array references. As explained in appendix A, this skewing allows us to interchange the loops without violating the dependencies of the




```
1: for x ← 0, X + 3T do
2:     for t ← 0, T do
3:         if max(t,x) < min(X + 3t, x + 1) then
4:             w ← x − 3t
5:             A[t + 1, w] ← step(A[t,w], A[t,w − 3], A[t,w − 2], A[t,w −
   1], A[t,w + 1], A[t,w + 2], A[t,w + 3])
6:         end if
7:     end for
8: end for
```

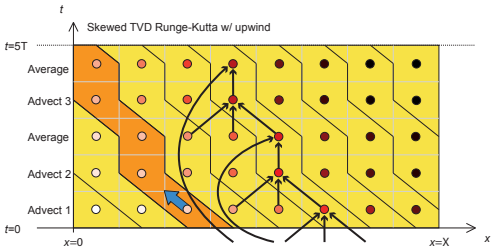
Figure 5 A number of steps on a 1D level set using the transformed traversal outlined in the pseudocode.

algorithm. Figure 4 shows the resulting traversal of the iteration space along with the transformed pseudocode. Iterations with references to the same entries of A are much closer together using this traversal, thus increasing the locality. If T is of the same magnitude as X, we have of course not achieved locality, and in general, loop skewing, loop interchange, *etc.* must be combined with tiling transformations as explained in section 3.2 [134, 135]. Briefly described, we separate the temporal dimension into tiles which are as large as allowed by main memory. These tiles are then executed in order.

For computations with higher order spatial schemes such as the variable third to fifth order accurate HJ WENO scheme, we can perform similar transformations to improve locality. The HJ WENO scheme employs a stencil of seven grid points, and thus each iteration depends on as many as seven previous results. However, this merely means that we have to skew the spatial dimension even more. In particular, the transformation $T_2 : [t,x] \rightarrow [t,x+3t]$ ensures that the loops can be interchanged. The resulting traversal order is shown in figure 5. Note how the "slope" of the skewed loops has changed to reflect the wider area of dependence of the stencil. Also note that the transformation used in this example would be perfectly "legal" in the previous example. The key observation is, that we seek the "legal" transformation that optimizes the slope, *i.e.* minimizes the *skew factor* of each loop as this provably minimizes memory reference costs [68]. In other words, we want to minimize the width of the wavefront of execution projected onto the x-axis.

The third order accurate TVD Runge-Kutta scheme for temporal discretization consists of five sub-steps in order to advance the solution one step forward in time [112]. More specifically it consists of two convex combinations of three Forward Euler time steps, and takes the following form for the general Hamilton-Jacobi equation $\frac{\partial \phi}{\partial t} + \mathbf{H}[\phi] = 0$:

$$
\begin{aligned}
\tilde{\phi}^{n+1} &= \mathbf{Euler}[\phi^n] \equiv \phi^n - \Delta t\, \mathbf{H}^n[\phi^n] \\
\tilde{\phi}^{n+2} &= \mathbf{Euler}[\tilde{\phi}^{n+1}] \equiv \phi^{n+1} - \Delta t\, \mathbf{H}^{n+1}[\phi^{n+1}] \approx \phi^{n+1} - \Delta t\, \mathbf{H}^n[\phi^{n+1}] \\
\tilde{\phi}^{n+\frac{1}{2}} &= \frac{1}{4}\left(3\phi^n + \tilde{\phi}^{n+2}\right) \\
\tilde{\phi}^{n+\frac{3}{2}} &= \mathbf{Euler}[\tilde{\phi}^{n+\frac{1}{2}}] \equiv \phi^{n+\frac{1}{2}} - \Delta t\, \mathbf{H}^{n+\frac{1}{2}}[\phi^{n+\frac{1}{2}}] \approx \phi^{n+\frac{1}{2}} - \Delta t\, \mathbf{H}^n[\phi^{n+\frac{1}{2}}] \\
\phi^{n+1} &= \frac{1}{3}\left(\phi^n + 2\tilde{\phi}^{n+\frac{3}{2}}\right)
\end{aligned}
$$




```
1:  for x ← 0, X + 3⌊(5T−1)/5⌋ + 2 do
2:      for t ← 0, 5T do
3:          x_start ← 3⌊t/5⌋ + min(t mod 5, 1) + ⌊(t mod 5)/3⌋
4:          if max(x, x_start) < min(x + 1, X + x_start) then
5:              w ← max(x, x_start) − x_start
6:              if t mod 5 = 0 then
7:                  A[t+1, w] ← step(A[t, w−1], A[t, w], A[t, w+1])
8:              else if t mod 5 = 1 then
9:                  A[t+1, w] ← step(A[t, w−1], A[t, w], A[t, w+1])
10:             else if t mod 5 = 2 then
11:                 A[t+1, w] ← average(A[t, w], A[t−2, w])
12:             else if t mod 5 = 3 then
13:                 A[t+1, w] ← step(A[t, w−1], A[t, w], A[t, w+1])
14:             else
15:                 A[t+1, w] ← average(A[t, w], A[t−4, w])
16:             end if
17:         end if
18:     end for
19: end for
```

Figure 6 An advection step on a 1D level set using the transformed traversal outlined in the pseudocode. Dependencies are shown for all the sub-steps of the advection step.

The separate sub-steps in the described method have different dependencies, *e.g.* the Euler steps have dependencies corresponding to the stencil used in the first example, while the convex combination steps only depend on earlier results on the same spatial position. These differences could be ignored, and we could apply a legal skewing transformation like T_1, but that would result in a suboptimal slope and width of the wavefront of execution, since the averaging steps do not require skewing in x. An optimal transformation must take this into account, and for a first order upwind scheme, we propose $T_3 : [t,x] \rightarrow [t, x + 3\left\lfloor \frac{t}{5} \right\rfloor + \min(t \bmod 5, 1) + \left\lfloor \frac{t \bmod 5}{3} \right\rfloor)]$, where $\lfloor\ \rfloor$ denotes the floor function. Figure 6 shows the resulting traversal of the iteration space, when the skewing has been combined with a loop interchange. Note that t is now used as a fictitious time variable such that $\left\lfloor \frac{t}{5} \right\rfloor$ denotes the time step and $t \bmod 5$ uniquely identifies one of the five assignment statements or sub-steps in the loop body. The described transformation goes beyond the framework of Wolf and Lam [134], and a careful analysis of the validity of the proposed transformation is provided in appendix A[2]. Transformations and code for TVD Runge-Kutta and BFECC combined with HJ WENO as well as generalizations to more spatial dimensions are also presented in appendix A.

3.1.2 Storage Mapping

Using the above transformations (combined with tiling) we have now achieved a wavefront of execution which permits locality. However, if each iteration writes to a separate entry in the expanded array A, the memory usage scales with the size of the level set grid. Therefore, we do not in practice store the entire temporal and spatial dimensions of A. Instead, each level set step streams an out-of-core grid as input and another as output while everything in between is stored in-core using a suitable storage mapping which maps each iteration to the address where it should store the value it produces [135]. The goal is to minimize memory usage, and at the very least ensure that it does not scale with the size of the level set grid. The storage mapping

[2]The analysis is performed on the algorithmically similar BFECC scheme, and the validity of the transformation for the TVD Runge-Kutta scheme is derived from that.

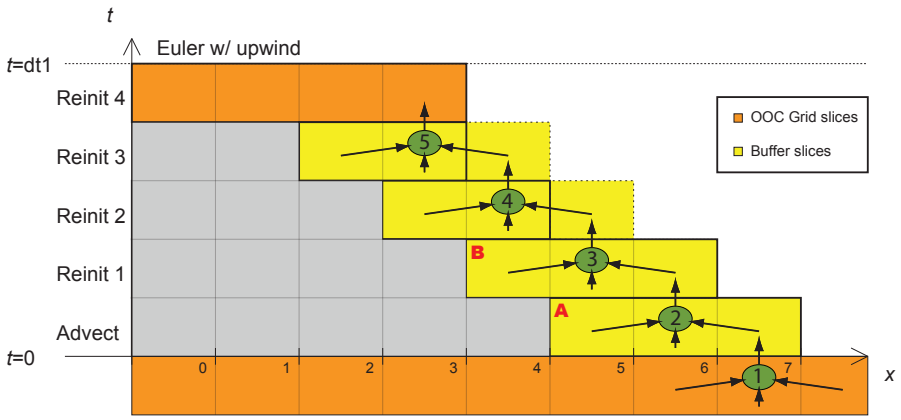


Figure 7 The storage requirements of a full level set step with advection and re-initialization steps using the Forward Euler scheme and upwinding. Yellow areas with dotted outlines indicate that a computation result replaces an entry in the buffer of temporaries corresponding to the same spatial position. The entry being replaced, which is not needed anymore by the computations on the wavefront of execution, is marked by a red letter.

applied in the first Euler example above (figure 4) is the trivial $\{[t,x] \rightarrow A[t+1,x-t]\}$, which is not independent of the size of the level set grid. On the other hand, we cannot just remove the temporal dimension of A and use the simple storage mapping of $\{[t,x] \rightarrow A[x-t]\}$, since *e.g.* storing the result from iteration $[0,1]$ in $A[1]$ overwrites a value on the wavefront of temporaries which is needed in later iterations such as $[0,2]$. Observing that the computations only depend on the results of the previous step, one improvement would be to only store the latest two rows of A and use the mapping $\{[t,x] \rightarrow A[(t+1) \bmod 2,x-t]\}$. This storage mapping still writes to a number of memory locations which scales with X, and thus the required memory does not fit in-core even though the wavefront of temporaries does. To improve this, we propose a storage mapping which is skewed in the same manner as the iteration space has been skewed. This idea leads to a small *buffer of temporaries* which only holds the wavefront of temporaries [135].

While the storage mappings of the previous paragraph lend themselves to be described by a simple formula, the skewed storage mappings which we propose are not as easily expressed in this formalism. Therefore we shall use diagrams to illustrate the storage mappings. This also provides the necessary intuition for code implementation.

In the following we present the mappings for some of the most popular level set discretization schemes. Figure 7 shows a "snapshot" of the execution of the Forward Euler scheme with upwinding, and in particular all the computations on the wavefront of execution. Orange areas indicate results which are read in from and written to out-of-core grids, while the yellow areas represent results stored in the in-core buffer of temporaries. The greyed out area indicates results from iterations that are not needed anymore, *i.e.* that are no longer part of the wavefront and buffer of temporaries. Finally, the yellow areas with dotted outlines indicate that the result stored there can replace one of the other entries in the buffer of temporaries corresponding to the same spatial position. These replaceable entries are marked by red letters. The green circles each

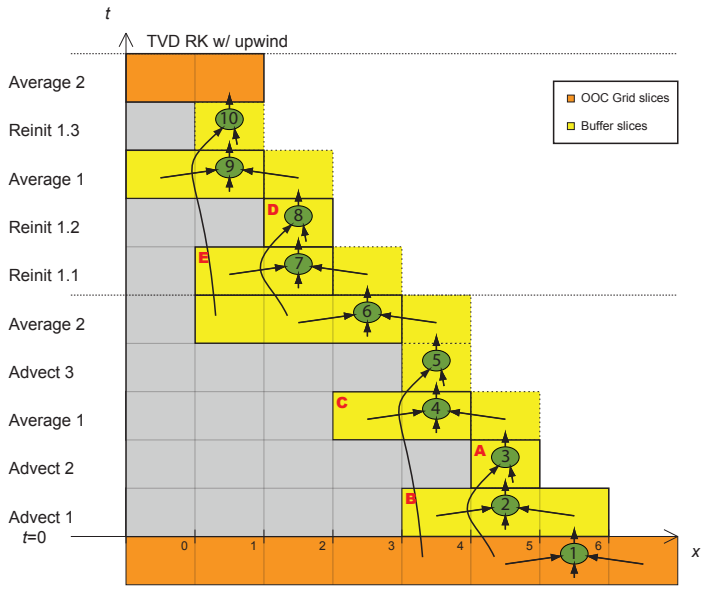


Figure 8 The storage requirements of an advection and a re-initialization step with the TVD RK scheme and upwinding. Note that the storage requirements are greater for subsequent steps than for the first step due to the averaging computations.

correspond to a computation and the black arrows indicate the dependencies. Two buffer entries above each other are needed to effectively propagate the wavefront of computations through the iteration space. Suppose the first computation (*i.e.* the circle with the number one) is about to advect entry 7 in the input out-of-core grid. It needs to store the result in the buffer of temporaries and because it cannot overwrite entries needed by the second through fifth computations, it stores the result as indicated. The second computation can similarly not overwrite needed values and therefore its result is stored as indicated. It should be noted that in one dimension it is possible that the second computation could use the entry in the buffer of temporaries directly to the left of it to store its result. However, it is important to stress that this does not generalize to N-dimensional level sets on DT-Grids. When we perform this generalization, the entries in the buffer of temporaries are in fact $(N-1)$-dimensional slices of varying sizes, so the result from the second computation would not necessarily fit.

Returning to Figure 7, we note that the third computation can write its result to the entry in the buffer of temporaries marked by the letter **A**, thus overwriting the entry which the second computation on the wavefront does not need anymore. Since this result is of the same size as the one overwritten, this does not pose a problem in N dimensions. Similarly, the fourth computation can write its result in the buffer entry marked **B**. This behavior generalizes to n advection/propagation and re-initialization steps, and the storage scheme works for computations of any stencil width w. The required size of the buffer of temporaries is given by $2r(n-1)+2$, where $r = \left(\left\lfloor \frac{w}{2} \right\rfloor\right)$ is the *effective radius* of the stencil.

Figure 8 shows a similar "snapshot" of the TVD Runge-Kutta scheme, again with first order upwinding. As can be seen from the grey and yellow areas with dotted

Temporal scheme	Buffer size
Forward Euler	$2r(n-1)+2$
TVD Runge-Kutta	$(7r+1)(n-1)+4r+2$
BFECC	$(6r+1)(n-1)+4r+2$

Table 1 The proposed memory requirements. r is the radius (or half the width rounded down) of the computational stencil employed, while n is the total number of steps in the level set step. Each entry in the buffer of temporaries is an $(N-1)$-dimensional slice.

outlines, we can immediately overwrite the result of the second computation (stored in the entry marked **A**) with the result of the first averaging computation (computation number 3), thus saving memory. Also, the fourth computation is able to write its result in the buffer entry marked **B**. Note that the storage pattern is slightly different for the second step, because the values written by the fifth computation cannot be overwritten as they are needed by the final averaging computation. The generalized formula for the size of the buffer of temporaries is given in table 1 along with the one for BFECC, which can be derived in a similar way. The proposed mappings result in memory requirements which scale with the size of the wavefront of execution rather than the size of the grid.

As mentioned, each entry in the out-of-core grids and the buffer of temporaries is actually an entire $(N-1)$-dimensional slice of the narrow band. In the one-dimensional examples above, the values are therefore just scalars, which the computations can operate on directly. In higher dimensions, we perform computations using sequential access on entire slices at a time. Furthermore, when dealing with more spatial dimensions, we tile the space and utilize the boundary grids mentioned in the overview. The specific storage mappings proposed for doing this will be explained in section 3.2.

3.1.3 The Fast Iterative Method

In section 3.1 we described how to solve the (pseudo time-dependent) re-initialization equation $\partial\phi/\partial t + S(\phi_0)(|\nabla\phi|-1) = 0$ to steady state in our out-of-core framework. In this section we describe how to apply a tiling scheme to the recently proposed *Fast Iterative Method* (FIM) [49] for solving the eikonal equation $|\nabla\phi| = 1$. This enables an out-of-core implementation that requires data to be streamed to and from memory only once. Furthermore, it can be combined with the skewing framework in section 3.1, hence requiring streaming to and from memory only once for all steps making up a level set iteration. The FIM has a number of properties which makes it well-suited for an out-of-core parallel implementation. First of all, it does not require a separate, heterogeneous data structure such as the heap required by the Fast Marching Method (FMM) [109, 127, 128]. Secondly, it can simultaneously update multiple grid points.

The FIM manages a list of active grid points which are iteratively updated until convergence. This *active list* is updated by adding and removing grid points based on a convergence measure. The main difference from the FMM is that grid points are updated independently and that the active list can move over grid points previously

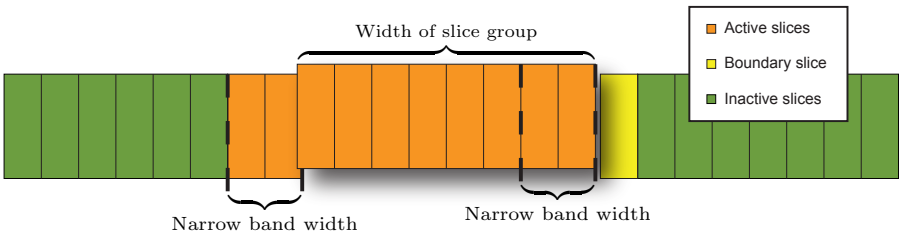


Figure 9 Our modified FIM processes the slices one group at a time. The active list can only propagate distance information within the orange slices. A band of slices from the previous group remain active to ensure correct backward propagation of information.

removed from the list and reactivate them as new information is propagated across the narrow band. This is in contrast to the FMM which maintains a heap of grid points sorted after their current distance to the zero-crossing and only removes them once they have the correct value. The consequence of the FIM's approach is that the grid points do not need to be updated in a strict order based on their distance to the zero-crossing as they can be reactivated.

To enable a streaming implementation, we modify the original algorithm by partitioning the level set slices into groups which are treated separately. In order to allow the correct propagation of distance information between the groups, a band of slices is shared between neighboring groups. The width of this band must be at least the same as the width of the narrow band. The motivation behind this is that, in the continuous case, the distance value at a certain point originates from points no further than 'distance' away from it. In our case the maximum distance computed equals the width of the narrow band. However, due to the seven-point star-stencils used for numerical computations, distance can, in the discretized case, travel infinitely on a sub-scale. This means that the distance value at a certain grid point may rely on the distance first being computed correctly at grid points further away than the width of the narrow band. However, as is evident from our numerical experiments in figure 2, the error introduced by limiting the width of the shared band of slices appears to be below the truncation error. Figure 9 shows three slice groups where the raised, middle group is currently being processed by our algorithm. The orange slices indicate the region in which the active list can compute and propagate distance information. We will call these the *active slices*. Notice that the last few slices of the previous group remain active such that new information can correctly propagate back to them. The yellow slice of the next slice group acts as a *boundary slice* and stores grid points which should have been included in the current active list. They are then used as the initial active list when treating the next group of slices. It should be evident from this description that the slice groups must be at least as wide as the narrow band.

We have experimented with two implementations of the active list. The main difference lies in the access pattern of the grid points in the list. The first implementation simply stores the points in a list and when processing them, it uses random access which is logarithmic in the number of connected components of the DT-Grid [90]. The second implementation utilizes a bit mask to determine which grid points are in the list. It then sequentially scans through all grid points in the current group and up-

Model	Reinitialization equation		FIM		Sliced FIM 1		Sliced FIM 2	
	$\lvert\cdot\rvert_\infty$	$\lvert\cdot\rvert_2$	$\lvert\cdot\rvert_\infty$	$\lvert\cdot\rvert_2$	$\lvert\cdot\rvert_\infty$	$\lvert\cdot\rvert_2$	$\lvert\cdot\rvert_\infty$	$\lvert\cdot\rvert_2$
	0.00452	0.000148	0.00697	0.000200	0.00697	0.000196	0.00697	0.000197
	0.0164	0.00251	0.0186	0.00203	0.0186	0.00200	0.0186	0.00200

Table 2 Error norms of the various FIM implementations on two different surfaces. Our sliced FIM implementations are as exact as the original algorithm. The slice group width is set to $2\times$ the narrow band width. For the algorithm from section 3.1, the initial ϕ is reset to $\pm\gamma$ away from the zero-crossing and the algorithm is run for 40 iterations using Forward Euler and Upwind differencing for the derivatives.

Algorithm	 $233\times167\times108$	 $985\times546\times657$
Reinit. eq.	5.9	61.8
Sliced FIM 1	4.4	49.4
Sliced FIM 2	6.8	108.6

Table 3 CPU times in seconds of the various FIM implementations. The examples were run with the same settings as in table 2.

dates the ones which are in the mask. This is done iteratively until convergence. We refer to these implementations as *Sliced FIM 1* and *Sliced FIM 2*, respectively.

Table 2 shows that the accuracy of Sliced FIM 1 and 2 is as good as for regular FIM. For both these examples, we used a slice group width of $2\times$ the narrow band width. For the algorithm from section 3.1, the initial ϕ was reset to $\pm\gamma$ away from the zero-crossing and the algorithm was run for 40 iterations using Forward Euler and first order upwind differencing for the temporal and spatial derivatives, respectively. Table 3 shows the running times for a few bigger examples. It is evident that the Sliced FIM 1 algorithm outperforms Sliced FIM 2 despite the slower access method.

3.1.4 REBUILD

To enable one or several complete level set steps to be implemented out-of-core in a single streaming pass, we must also consider how to adapt the narrow band rebuild algorithm of the DT-Grid. Similar to the substeps of computation in the TVD RK algorithm in figure 6, the substeps of the rebuild algorithm can be skewed to maximize locality. We refer to the original DT-Grid paper for full details on the rebuild algorithm [90], and present here only a simplified version. In particular the rebuild algorithm consists of the following steps, where we assume ϕ to be a signed distance function, γ to be the Euclidean width of the narrow band, N to be the dimension, and H to be the width of the dilation measured in grid cells (the rebuild process is illustrated in the rightmost part of figure 10 with $N = 1$ and $H = 1$):

1. Copy grid points with $|\phi| < \gamma$ to an intermediate grid.

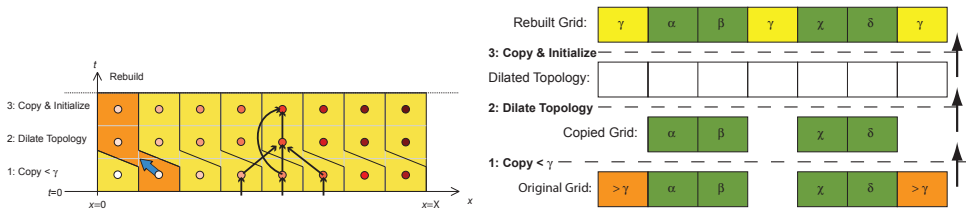


Figure 10 Leftmost: Illustrates the skewed iteration space of the rebuild process with the wavefront of execution outlined in orange. The black arrows indicate the dependencies. **Rightmost:**Illustrates the steps involved in the rebuild process of a 1D DT-Grid. To emphasize the relationship with the skewed iteration space (left), the figure should be read from the bottom and up. In this example the original grid consists of two separate connected components. Grid points with numerical value less than γ are shown in green. Values larger than γ are shown in orange and new grid points are shown in yellow.

2. Dilate the topology of the intermediate grid with a stencil shaped as a hypercube of dimensions $(2H+1)^N$. This may change the topology of the grid. Then allocate uninitialized storage for the values of the grid.

3. Copy values of grid points that exist in the intermediate grid to the final grid, and initialize new grid points to a numerical value of γ.

In the original paper each of these steps are completed before the next commences. However, as illustrated in the leftmost part of figure 10 the wavefront of execution can be made independent of the grid dimensions. In particular steps 1 and 3 require no skewing since a computation at these substeps depends only on the computation immediately below in the original iteration space. Since step 2, the dilation step, utilizes a hypercube-shaped stencil for dilation, a minimal skewing of H grid points is required in each spatial dimension at this level of computation. When including a rebuild step on the wavefront of execution, where both the prior and the subsequent steps operate in the buffer of temporaries as opposed to out-of-core, the number of entries in the buffer of temporaries is increased by $(2H+1)+(2\lfloor\frac{w}{2}\rfloor+1)$ for a simulation employing forward Euler. In particular step 1 requires $2H+1$ additional entries corresponding to the width of the dilation stencil, and steps 2 and 3 require $2\lfloor\frac{w}{2}\rfloor+1$ additional entries corresponding the width of the propagation stencil. Similar arguments hold for a simulation employing TVD RK or BFECC time stepping thereby obtaining $(2H+1)+(3\lfloor\frac{w}{2}\rfloor+1)$ additional entries. If the result of the rebuild step is placed out-of-core, the number of entries is only increased by $(2H+1)$, independent of the time stepping approach, since there is no propagation stencil.

3.1.5 Concatenating Multiple Level Set Steps

In the the previous sections, we have only dealt with performing one level set step during a single pass over the level set. If the amount of main memory allows it, several level set steps can be concatenated into one pass. Figure 11 shows a concatenation of two level set steps with advection and re-initialization steps (for simplicity the rebuild

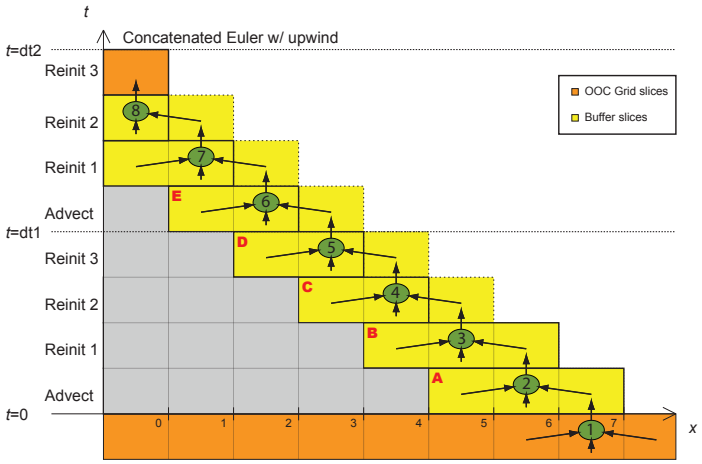


Figure 11 Two concatenated Euler level set steps. Notice how all intermediate values are stored in-core while out-of-core grids are used only as input and for the final output. Also note that we have abstracted away the sub-steps of the rebuild step for simplicity.

steps have been omitted). Note how the intermediate results of the first level set step are kept in memory as opposed to in figure 7. This lowers the disk traffic and lessens the load on the bandwidth to the disk, which facilitates using slower disks and/or faster processors with no performance penalty as well as running several simulations or computing on several tiles at the same time.

This approach requires that a time step size is chosen for some level set steps in the simulation *before* the previous steps have been completed, which is not always possible. In most cases, however, one can perform conservative estimates of or exactly determine the allowed time step size. This is the case for *e.g.* mean curvature flows where the latter is possible and for analytical flows where only the former is an option. One notable exception is fluid simulation flows where a guaranteed stable estimate can not be made.

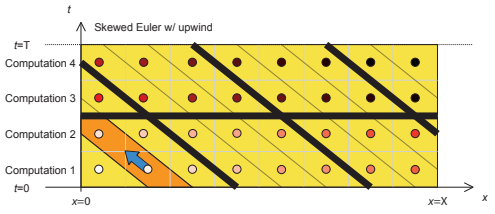
Even when performing only a single level set step per pass, one does not want to perform a separate pass over the data to determine the time step size for the next step. If the velocity field for the next step is available, one can combine the rebuild algorithm with an evaluation of the field for all grid points in the new narrow band and thus determine the allowed time step size.

3.2 Tiling

In this section we first describe the tiling of the skewed iteration space and next explain the storage mapping that tiles the actual data layout.

3.2.1 Tiling the Iteration Space

Whereas skewing of the iteration space is required to facilitate permutation of the iteration directions, tiling is in general required to ensure locality of references to data.




```
1: for t2 ← 0, T, BT do
2:     for x2 ← 0, X+T, BX do
3:         for t ← t2, min(T, t2+BT) do
4:             for x ← max(t, x2), min(X+t, x2+BX) do
5:                 A[t+1, x−t] ← step(A[t, x−t], A[t, x−1−t], A[t, x+1−t])
6:             end for
7:         end for
8:     end for
9: end for
```

Figure 12 Leftmost: Shows the geometric outline (thick black lines) of the *skewed* tiles resulting from tiling the iteration space. In this case a tiling of $B_T = 2$ was used in the temporal dimension and a tiling of $B_X = 3$ was used in the spatial dimension. The wavefront of execution which is independent of the grid dimensions is shown in orange in the lower left tile. **Rightmost:** Shows the code corresponding to the iteration space traversal on the left.

This is caused by the fact that along a direction in the iteration space we may reference more data than can fit into memory. Hence, even though later computations reuse data items, omitting skewing will in the worst case have to load them into memory every time they are referenced. Tiling, or blocking, is conceptually simple and illustrated in 1D for a first order spatial method combined with Forward Euler in figure 12. The iteration along each direction of the iteration space is simply divided into tiles of equal size by splitting the corresponding loop into two loops, where the loop that steps over the tiles is called the controlling loop. As shown in the rightmost part of figure 12 tiling is combined with loop interchange to place the controlling loops as the outermost loops in order to ensure locality. More concretely, if a given level set simulation takes T substeps then without skewing and tiling, data has to be streamed to and from memory T times. By introducing skewing and a tile size of B_T in the temporal direction, the number of times data has to be streamed is reduced to T/B_T. In order for this to work, the size of the wavefront of temporaries described in section 3.1 must fit into memory. This poses restrictions on how large B_T can be and additionally a tiling in the spatial dimensions may be required as well. In fact, for a simulation on a N-dimensional grid, tiling is required in at most $N-1$ spatial dimensions in order to make the wavefront of execution independent of the grid dimensions. Consider the 1D example in figure 12. In this case it is not necessary to tile in the x-direction to ensure locality because the dimensions of the wavefront of execution is independent of the spatial direction in which it travels (in this case the x-direction). For the 3D simulations considered in this article, we at most tile in the y- and z-directions and always leave the x-direction untiled. Note that the choice of untiled direction is arbitrary.

3.2.2 Tiled Storage Mapping

Skewing and tiling the iteration space optimizes locality for level set computations. However, these transformations are not sufficient to yield a fast practical implementation. Recall that in order to obtain computational and storage efficiency we implement our proposed framework on the DT-Grid data structure. Accessing data in tiles on a DT-Grid will result in a large number of non-sequential access operations that have logarithmic time complexities. In an out-of-core implementation the non-sequential access operation furthermore results in a logarithmic IO complexity as well as disk search operations which are expensive operations relative to sequential disk access.

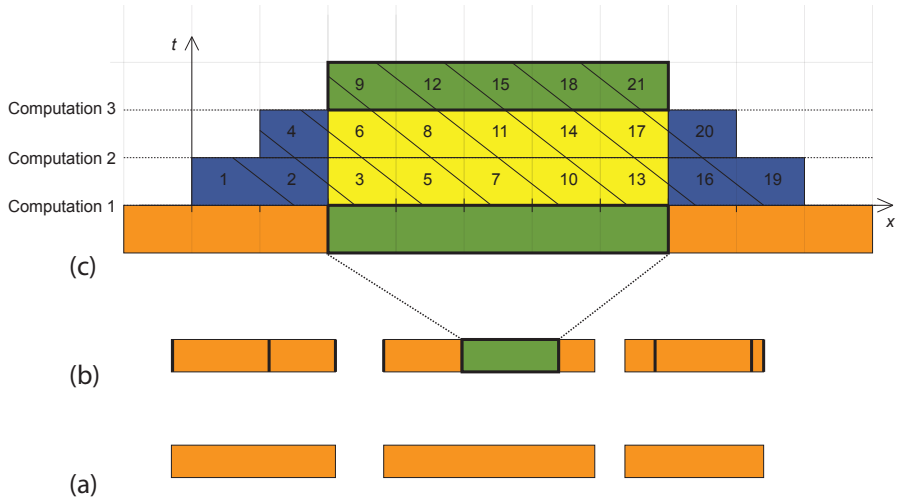


Figure 13 **(a)** Untiled 1D grid consisting of three connected components. **(b)** The grid in (a) tiled using a fixed tile size. Note that due to the connected components the part of the narrow band inside each tile may vary in size. **(c)** The enlargement of a single tile, shown in green. The boundary data from neighboring tiles required for three complete computations on the tile is shown in orange. The values allocated as temporaries are shown in blue and yellow. Note that only the wavefront of temporaries are stored in-core at any time during the simulation. The output from three complete computations on the tile is shown in green at the top. The wavefronts are indicated by the skewed lines, and the exact order of computations is indicated by the numbering.

This suggests that in order to obtain feasible run times a transformation of the layout of data on secondary storage is required in addition to iteration space tiling. In particular, a tiling of the narrow band is required. Figure 13.a shows a 1D un-tiled grid, and figure 13.b shows a tiled version of the same grid, where each tile is stored as a separate grid. The tile boundaries of the grid correspond to the tile boundaries of the iteration space. Inside each tile, computations are skewed with respect to time and temporary storage allocated as described in section 3.1 and illustrated in figure 13.c. In order to complete several iterations inside each tile, boundary data from neighboring tiles is required which is also illustrated in figure 13.c. A straightforward way to do this would be to also stream parts of neighboring tiles through memory when performing computations on a specific tile. However, this is infeasible due to the aforementioned penalties of random access, and further impeded by the fact that large page sizes are used when transferring data from disk to memory. Thus we may in the worst case end up streaming all of the neighboring tiles through memory when in fact only a relatively thin band of boundary data is needed. The situation becomes more intractable for higher dimensional grids, *e.g.* in 3D we may end up streaming the grid to and from memory nine times if tiling in two spatial dimensions. The solution to this is to store separate boundary grids that contain only the boundary data needed to complete the computations inside each tile. Note that the width of the boundary grids is equal to the width of the wavefront of execution. In particular, a wavefront including N Forward Euler fictitious time steps has a width of $(N-1)\lfloor \frac{w}{2} \rfloor$, whereas a wavefront including

M TVD RK or BFECC fictitious time steps has a width of $(M-1)(w-1)$, where w is the width of the stencil. Each rebuild step included in the wavefront adds one to its width. Note that the width of the boundary grids is proportional to the number of fictitious time steps. In all of our simulations the boundary grids are very small compared to the size of the tiles, as B_X is orders of magnitude larger than B_T, where B_X and B_T are the tile sizes in space and time respectively. The reason for this is that B_T in most cases only includes a single level set step. If several level set steps are concatenated as discussed in section 3.1.5, there is a tradeoff between lowering the requirements on IO bandwidth and the increased overhead arising from boundary grids. We call the resulting data structure the *Tiled DT-Grid* and an example in 3D was depicted in figure 1. The tiled DT-Grid consists of three components: A *coarse grid*, several *tile grids* and several *boundary grids*. Each cell in the coarse grid corresponds to a tile in the spatial dimensions. Since we always leave the x-direction untiled, the spatial extent of such a cell will be infinite in at least one direction. A cell in the coarse grid essentially holds pointers to a single tile grid and to a boundary grid for each boundary element (in 3D either an edge or a face) along which two cells are adjacent. Hence for a 3D grid the number of boundary grids for each cell will vary between zero and eight; zero if no tiling is applied and eight if tiling in the y- and z-directions since there will be eight neighboring grids; one along each of the four faces and one along each of the four edges of the tile (see figure 2). Both the coarse grid, the tile grids and the boundary grids can be stored separately as out-of-core DT-Grids. Since the intersection of the narrow band with the boundaries of a tile may result in a level set that is not closed, special algorithmic care has to be taken for a DT-Grid implementation, and how to handle iteration with a stencil and narrow band rebuild is described in [89].

Concurrent with computations on a specific tile, the grid points of the tile grid itself as well as the grid points of adjacent boundary grids from neighboring tiles are merged into a single DT-Grid when streamed into memory. This is facilitated by the lexicographic storage order of the underlying DT-Grid storage used for both tile and boundary grids. Hence the grid points are sorted with respect to the lexicographic order, and the merging can be implemented using a heap which contains at most as many elements as there are boundary grids plus one. The heap adds an overhead compared to streaming alone, however optimizations are possible in our case. Firstly, the number of entries in the heap is limited by the number of boundary grids plus one (at most 5 in our case). Secondly, merging only needs to be invoked (and hence the heap activated) at the boundaries between the tiles and the boundary grids, where the origin (tile or boundary grid) of grid points changes.

Notice from figure 13.c that computations on grid points that fall inside a boundary grid will be performed twice on each tile if processed independently. The computational overhead will be proportional to the narrow band volume inside the boundary grids. Generally this will be small compared to the narrow band volume inside the tile grids, and this strategy also has the advantage that each tile can be processed in parallel. If duplication of computations on boundary grids is not desirable, the tiles can be processed sequentially and computations performed on a boundary grid can be stored to disk temporarily and used for initialization in adjacent tiles.

4 Results and Discussion

4.1 Single Threaded Performance

In this section we present benchmark evaluations of our out-of-core level set framework and compare its throughput to the throughput of the out-of-core framework by Nielsen *et al.* [91] as well as to the throughput obtainable for in-core simulations on the original DT-Grid data structure [90].

For the benchmark tests we have employed the following methodology. We consider three different level set flows: (1) Constant normal propagation (exemplified by an erosion), $\phi_t - |\nabla\phi| = 0$. (2) Advection in a velocity field where the maximal velocity is known prior to each level set step (exemplified by a translation), $\phi_t + \vec{V}\cdot\nabla\phi = 0$. (3) Mean curvature flow, $\phi_t - \kappa|\nabla\phi| = 0$. Each of these flows are simulated using two different numerical schemes: (1) A forward Euler temporal discretization combined with a spatial discretization of first order one-sided differences for the hyperbolic terms and second order central differences for the parabolic terms. (2) A third order TVD RK discretization in time combined with a spatial discretization of HJ WENO for the hyperbolic terms and second order central differences for the parabolic terms. We evaluate each combination of flow and numerical scheme on a narrow band DT-Grid based level set representation of the Stanford Bunny (see figure 1) in increasing resolution. For the high order numerical scheme (2), the maximal thickness of the narrow band corresponds to 7 grid cells, and for the low order numerical scheme (1) a thickness of 4 grid cells is used. For numerical scheme 1, we have run the simulations on narrow band grids ranging over effective resolutions roughly 1000^3 (0.085GB) to 14000^3 (17GB). For numerical scheme 2, the effective resolution of the narrow band grids range from 1000^3 (0.13GB) to 10000^3 (14 GB). In each case the highest resolution corresponds roughly to the narrow band that allows for the maximal number of degrees of freedom representable in 32 bit. Since the narrow band is thicker for numerical method 2, the largest model employed here is smaller in terms of resolution. During simulation roughly twice the storage is required, since an input and an output out-of-core grid must be represented simultaneously. The simulations were run on a computer with 32Bit Windows XP Pro, a single-core 2.41GHz AMD CPU, 1 GB of memory and a 10000 RPM Western Digital Raptor disk. The benchmark programs were implemented in C++ and compiled using Visual Studio 2005 with maximal optimization enabled. In our implementations of the framework proposed in this paper and the framework in [91], only the code for the level set algorithms differed. The underlying DT-Grid as well as the prefetching and page-replacement schemes all utilized the exact same code. For the single threaded benchmarks we utilized only a single tile and hence no boundary grids.

4.1.1 Performance of Skewed Simulations

As illustrated in figure 14, the performance of the DT-Grid drops notably as the main memory limit is reached. The out-of-core framework in [91] improves the performance, but remains IO limited as shown in figure 15 left. In contrast the framework

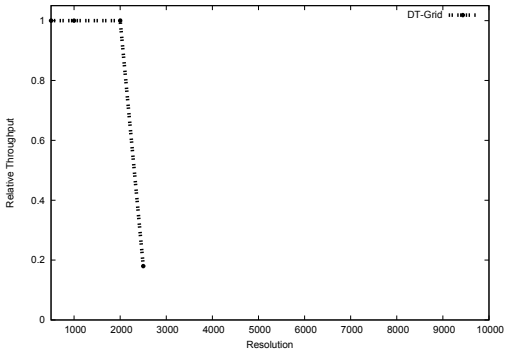


Figure 14 The performance of the in-core DT-Grid drops when the main memory limit is exceeded.

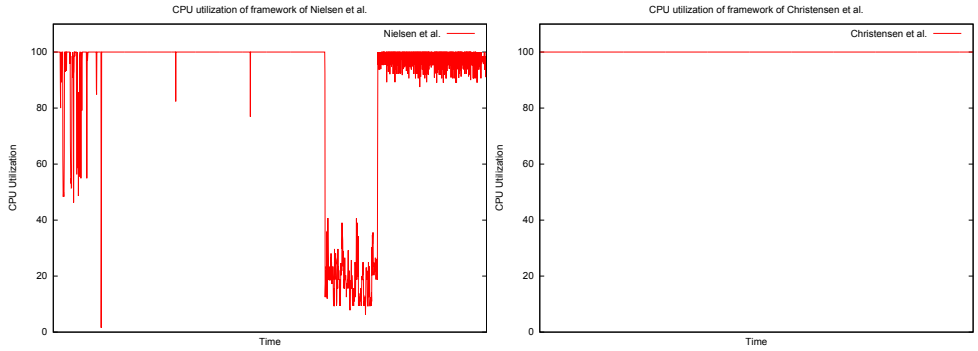


Figure 15 This figure shows the CPU utilization over the course of a single time-step of an out-of-core simulation on the Stanford Bunny at resolution $\approx 8000^3$. **Left:** The framework in [91] is IO limited for both low and high order methods. **Right:** Our framework utilizes the CPU 100% and is thus CPU limited although the data is streamed to and from memory.

proposed in this paper is CPU limited as depicted in figure 15 right.

Results from the benchmarks are shown in figure 16. For numerical scheme 1 there seems to be the following tendency: For our proposed framework, the ratio of throughputs (number of processed grid points per second) is roughly constant and appears to rise slightly for the propagation and advection tests. In all cases, our new framework outperforms that in [91]. In particular the performance of the framework in [91] drops as the resolution is increased, a consequence of an increasing IO bottleneck. We expect the performance to converge asymptotically to some fixed ratio of throughputs not reached within the resolutions spanned by our tests.

For numerical method 2, performance is roughly constant for both frameworks, but our framework outperforms that in [91]. The reason for this is that the framework in [91] is IO limited. Contrary to the case of numerical method 1, the ratio of throughputs has in this case converged for the framework in [91] . Performance is higher in the case of numerical method 2 than numerical method 1. This is due to the fact that numerical method 2 is of higher order, and the CPU overhead associated with our

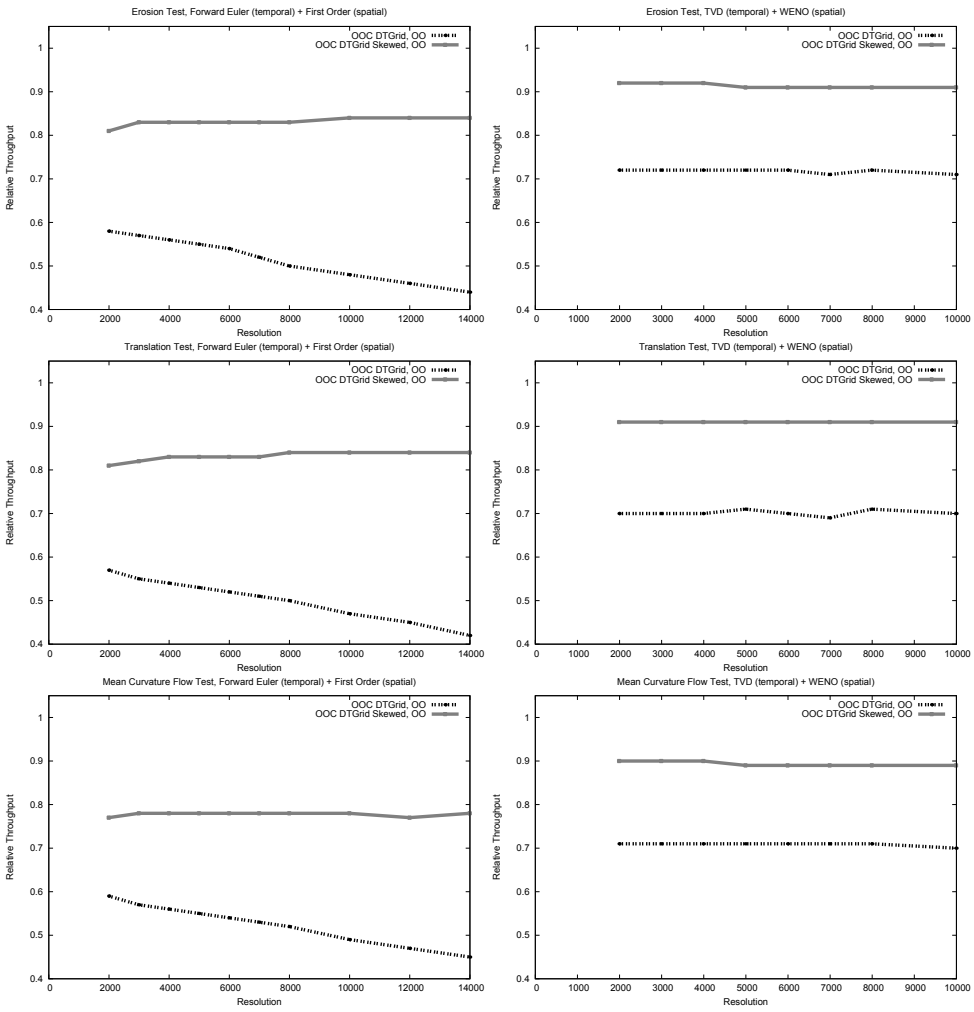


Figure 16 Benchmark results from a normal propagation test (top), advection test (middle) and mean curvature flow (bottom). Results are reported as the ratio of throughputs (number of processed grid points per second) obtained by our framework (OCC DTGrid Skewed, OO) and the framework in [91] (OOC DTGrid, OO) to the throughput obtained by an in-core DT-Grid simulation at 1000^3. The left column shows results from numerical scheme 1 and the right column shows results from numerical scheme 2, as explained in section 4.1.

proposed framework is constant per byte streamed to and from disk. Since the higher order schemes require more CPU time, the relative overhead of our framework is lower for a higher order than a lower order method.

We conclude that for both numerical methods, the throughput of our framework appears to be sustained, independently of resolution and numerical method.

As we have argued above, the current framework is CPU bound with an overhead between $8-23\%$ compared to the DT-Grid in-core narrow band level set method. The question is if this overhead can be reduced further. From analyzing the framework, it appears that about two thirds of the introduced overhead arises from checks in the

code that ensure that iterators are updated correctly whenever they move to data in a new disk page. Because these checks have to be done for each access into the topology and value data, they comprise a substantial part of the overhead compared to a fully in-core method which does not have to perform these checks. We believe that most of the overhead associated with these checks can be eliminated on a 64 bit operating system, if the narrow band level set is not larger than the virtual address space. The strategy is to essentially memory map the file into the virtual address space, but in such a way that only active pages are actually allocated and that pre-fetching and page-replacement is done using the methods in [91]. We are currently investigating this.

4.2 Multi Threaded Performance

Parallellization techniques are required in order to run simulations within feasible time constraints. In this section we evaluate the performance of our framework in a multi-threaded environment. In particular we evaluate its performance when running several out-of-core simulations on the same disk and compare performance to the out-of-core framework by Nielsen *et al.* [91]. Furthermore we evaluate the parallelization overhead introduced by combining skewing and tiling transformations and benchmark the parallel performance of our framework. For single threaded applications we have not found it necessary to tile the grid in practice. The reason is that the computation-time becomes infeasible before the memory limit is exceeded by the number of slices required in-core by the computation. For this reason we only apply tiling for multi-threaded applications.

The multi-threaded experiments were run on a computer with 64Bit Windows Vista Business, two Intel Xeon 2.8GHz quadcore processors (8 cores in total) and three 7200 RPM disks. All experiments utilized less than 2GB of memory. Our framework was parallelized using Intel Thread Building Blocks [104], and boundary grids were kept in-core for these tests.

4.2.1 Performance of Multiple Simulations on the Same Disk

In this experiment we compare the performance of the framework in [91] to our framework in the situation where several concurrently running simulations utilize the same disk. As test case we consider mean curvature flow on the Stanford Bunny at resolution $\approx 8000^3$ combined with numerical method 1 described in section 4.1. In our framework we concatenated two level set steps as described in section 3.1.5 to obtain better IO-efficiency. Figure 17 shows the result of the experiment. The framework of Nielsen *et al.* is IO-limited for a single simulation, and the throughput drops significantly as the number of simulations utilizing the same disk is increased. On the contrary, the throughput of our framework by and large stays constant, except for a slight drop ($\approx 1\%$) in the beginning.

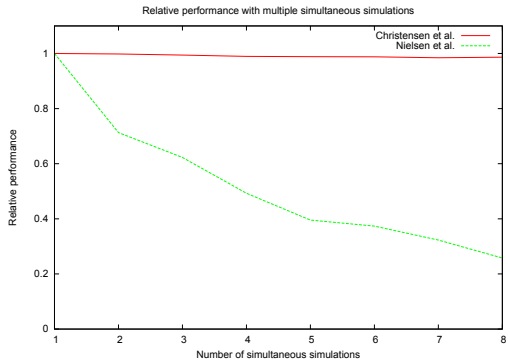


Figure 17 This figure shows the relative throughput (defined as actual throughput divided by throughput when only a single simulation is utilizing the disk) as a function of the number of simultaneous simulations on the disk.

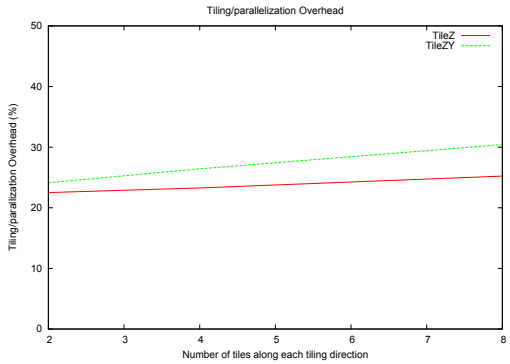


Figure 18 This figure shows the tiling/parallelization overhead of our framework as a function of the number of tiles along *each* tiling-direction. **TileZ** tiles only in the Z-direction whereas **TileZY** tiles in both the $Y-$ and Z-directions. The tiling/parallelization overhead is defined as percentage increase in execution time resulting from a single-threaded run of our framework using both tiling and skewing versus a single-threaded run of our framework using only skewing transformations. As test case we consider advection of the Stanford Bunny at resolution $\approx 8000^3$ combined with numerical method 1 described in section 4.1.

4.2.2 Parallelization Overhead

The best serial algorithm is seldom the best parallel algorithm [104], and a parallelization overhead is introduced by the tiling transformation which in turn is required in order to facilitate multi-threading. For a given dataset, the overhead grows as the number of tiles grows, as this will cause an increase in the number of boundary grids and hence redundant computations. Additionally, the overhead depends on the size of the boundary grids which depends on the number of concatenated level set steps as well as the size of the numerical stencils. Finally the cost of merging grid points from boundary grids and tiles will increase since the heap used to sort these will contain more elements. Figure 18 shows the tiling/parallelization overhead for two different blocking schemes. The overhead in this case lies between 23% and 30% when comparing

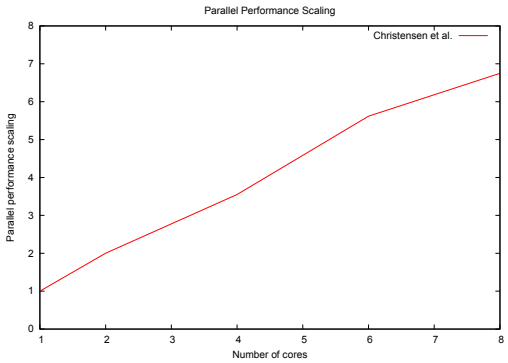


Figure 19 This figure shows the parallel performance scaling as a function of the number of cores. To emphasize the scaling trend, the parallel performance scaling is computed as the execution time divided by the execution time of the parallel version of our framework (using both skewing and tiling) running on a single core. As test case we consider advection of the Stanford Bunny at resolution $\approx 8000^3$ combined with numerical method 1 described in section 4.1.

single-threaded performance. A conclusion to be drawn from figure 18 is that tiling in only one direction is preferable over tiling in two directions, if memory requirements permit it.

4.2.3 Performance of Skewed and Tiled Simulations

Once the simulation is set up, there are virtually no serial sections in our framework implementation, except for the code that logs performance and saves level set data to disk. Hence according to Amdahl's law, our framework should have good theoretical parallel performance scaling properties. As can be seen from figure 19, the parallel performance of our current framework implementation scales sub-optimally as the number of cores are increased, obtaining a parallel speedup of 6.75 using 8 cores and using roughly 104 seconds per level set time step. The reason for not obtaining $8X$-performance is not due to IO limitations, since our framework remains CPU limited. In fact we observed similar scaling properties when keeping all components (values and topology) in-core on smaller data sets. For the tests in this section we also attempted to diminish load-imbalancing by applying a simple non-uniform tiling strategy in which the number of tiles equals the number of cores used for the computations. Each core is then assigned to a specific tile, and the tiles are constructed in such a way that the number of grid points in each tile is roughly constant. Furthermore the data was distributed evenly on the computer's three disks. We leave an investigation of using more cores, a different architecture as well as improvements of the parallel performance scaling of our framework for future work.

5 Applications

5.1 The Divergence-Free Advection Test

In this section we demonstrate an extreme level set deformation by advecting 8 spheres through the incompressible, periodic velocity field originally proposed in [67, 23]:

$$u(x,y,z) = 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos\left(\frac{t\pi}{T}\right)$$
$$v(x,y,z) = -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos\left(\frac{t\pi}{T}\right)$$
$$w(x,y,z) = -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos\left(\frac{t\pi}{T}\right)$$

where $T = 3$ is the period of t. The velocity field is reversed at $t = 1.5$, and the advected level set should return to its original shape at time $t = 3$. The spheres have radius 0.125 and are placed in a unit computational domain at positions $(0.15, 0.15, 0.85)$, $(0.15, 0.85, 0.15)$, $(0.35, 0.35, 0.35)$, $(0.35, 0.65, 0.65)$, $(0.65, 0.35, 0.65)$, $(0.65, 0.65, 0.35)$, $(0.85, 0.15, 0.15)$ and $(0.85, 0.85, 0.85)$. The advection equation $\partial\phi/\partial t + \nabla\phi \cdot (u,v,w) = 0$ where ϕ is the level set function and t is time was solved using a third order accurate TVD RK discretization in time and a three – fifth order accurate HJ WENO discretization in space. Figure 20 shows the surfaces at various times during the deformation. The unit computational domain is sampled at resolution 2048^3, and the DT-Grid narrow band contains approximately 85.7 million voxels at the beginning. It takes up 0.37GB of storage and roughly twice as much is needed during simulation. At $t = 1.5$ the storage requirements peak at 1.4GB and approximately 343 million voxels are contained in the narrow band. The simulation was run on a Mac Pro work station with two Intel Xeon quad core 2.80GHz CPUs, 4GB of memory and 4 7200 rpm hard drives, which were all utilized. Tiling was performed in the Z-direction with a CPU assigned to each tile, and each iteration took from 103 seconds in the beginning and end ($t = 0$ and $t = 3$) to 390 seconds around the peak ($t = 1.5$). The rendering was performed by in-core ray tracing on a different machine.

5.2 Mean Curvature Flow of Surfaces

The applications of curvature-based surface flows are vast. In this section we illustrate the use of mean curvature motion for surface smoothing expressed by the simple level set equation $\partial\phi/\partial t = \kappa|\nabla\phi|$, where κ is the mean curvature of the surface, ϕ is the level set function and t is time. We discretized the equation using first order accurate Forward Euler in time and second order accurate central differences in space. Figure 21 shows the Lucy statuette from the Stanford Scanning Repository [115] scan converted into a DT-grid narrow band level set in resolution $17149 \times 9987 \times 5734$ using the method for manifold meshes proposed in [47]. Each instance of the DT-Grid is 11.5GB, and during simulation roughly twice the storage is required. The simulation was run on a Dell Inspiron 8600 Laptop with 1GB of memory and a 7200 rpm hard drive. Even on this hardware configuration, our out-of-core framework remains CPU limited, one iteration taking approximately 108 minutes. While not necessarily feasible for simulations requiring many iterations, due to time constraints, this illustrates

that our framework is applicable even on architectures with limited resources. Notice that in the initial surface on the left, the triangulation of the original Lucy polygonal model, from which the level set was scan converted, is visible. However, since the model is super-sampled using the level set, and hence has a larger number of degrees of freedom, the mean curvature motion effectively results in a smooth interpolation of this model to higher resolution. The rendering was done by Gouraud shading of triangles, streamed to the graphics card and extracted from the level set using the marching cubes algorithm [71]. Since the marching cubes algorithm will produce many more triangles for this 11.5GB level set than can be stored on the graphics card, a draw command is repeatedly issued whenever a fixed number of triangles have been streamed to the graphics card.

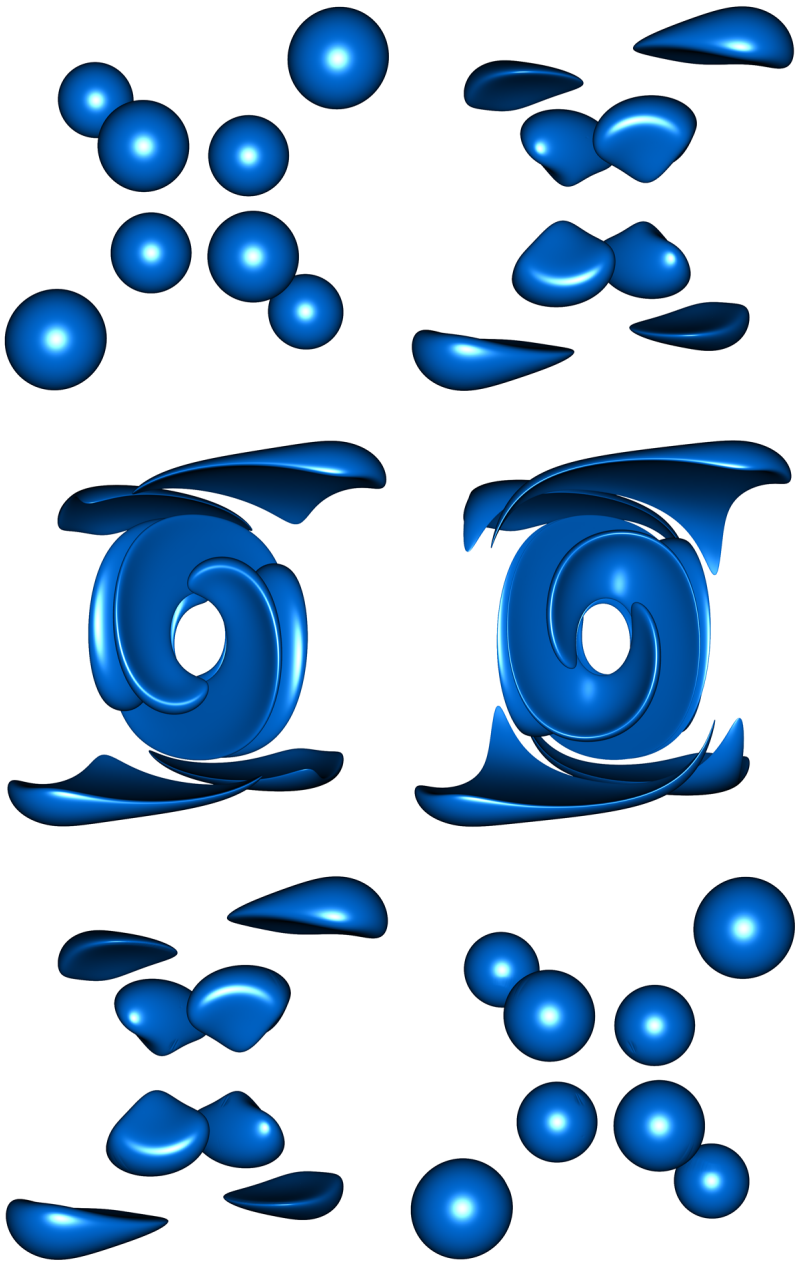


Figure 20 Results of the divergence-free advection test on 8 spheres in resolution 2048^3 requiring up to 1.4GB of storage. At the peak ($t = 1.5$), the narrow band contains approximately 343 million voxels. From top left to bottom right the images are from $t = 0$, 0.3, 0.8, 1.5, 2.7, 3.

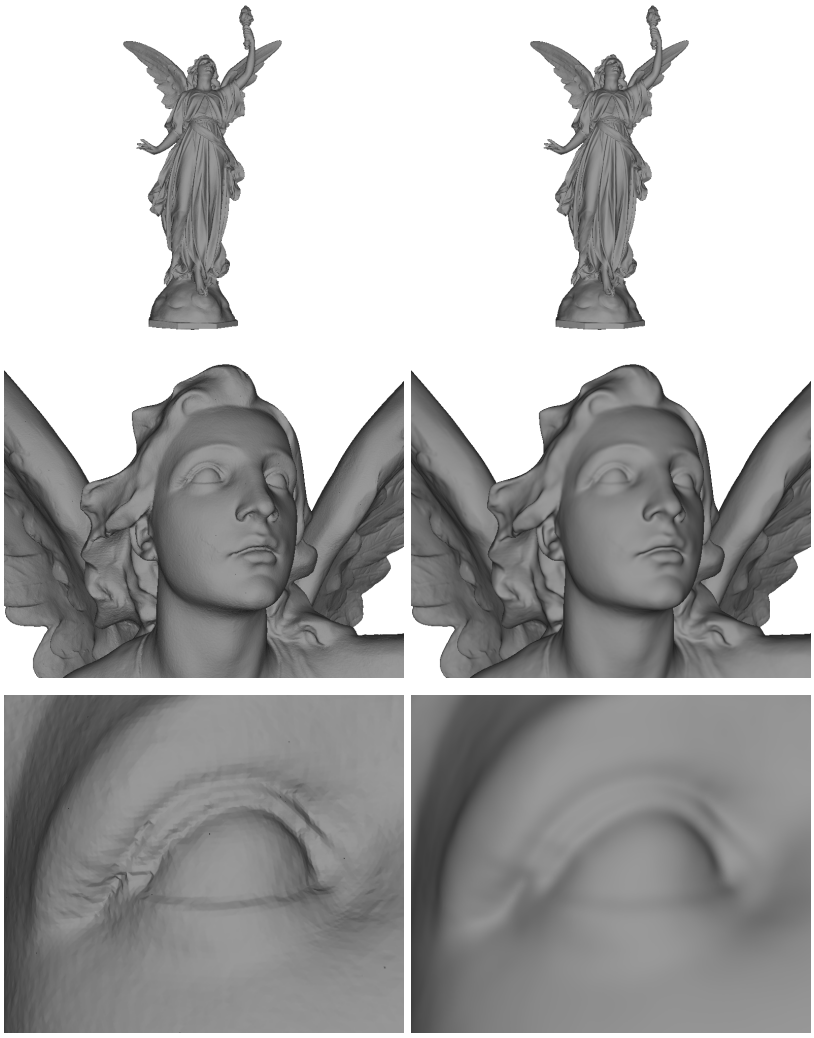


Figure 21 Results of the mean curvature motion on the lucy statue scan converted into a DT-grid in resolution $17149 \times 9987 \times 5734$ taking up 11.5GB of storage. Initial surface to the left and after 200 iterations to the right. The first row shows the whole statue, the second a zoom in on the head region and the third depicts an even closer zoom to the eye.

6 Conclusion and Future Work

We have presented a fast, storage efficient and parallelizable out-of-core framework for performing computations on level sets at resolutions only limited by the size of secondary disk space. The framework utilizes code transformations to allow the combination of multiple passes over the data into a single pass. As a result, the level set algorithms become CPU limited and maintain a throughput of up to 92% of in-core performance, independently of the level set resolution.

The framework still incurs an overhead compared to a strictly in-core level set method. As mentioned in section 4.1.1, we believe that a careful engineering effort can reduce this overhead. In the future, we wish to further investigate and improve the performance of our out-of-core framework for parallelization and multi-threading. In particular, given the high resolutions enabled by our framework, computation times are now the main bottleneck in achieving results at a desired resolution within a desired time frame. Investigating our framework in the context of parallelization over more cores and CPUs as well as in combination with implicit methods for solving PDEs, which would allow for larger time-steps, are promising directions for future work. Additionally, an automatic determination of the number of computations that can be concatenated in order to fully utilize internal memory, would ensure that no excess disk bandwidth is used. It would probably require an initial pass through the data coupled with a statistical model of the upcoming memory requirements resulting from the computation. Similarly, determining and adjusting tiling-directions as well as tile sizes automatically depending on the amount of memory available, would be useful. Currently, we assume it is possible to set up a configuration at the beginning that will remain valid throughout the lifespan of the simulation. This has been the case for all the simulations presented in this paper, however the ability to adjust the tile sizes dynamically could prove useful for load balancing the CPUs.

While we have focused entirely on level set computations, our techniques are also relevant for general out-of-core stencil based computations, and it would be interesting to investigate similar strategies for *e.g.* fluid simulation.

Acknowledgements. The authors wish to thank Ola Nilsson for help with figure 1. This work was partially funded by the Danish Agency for Science, Technology and Innovation.

A Data Locality Analysis

In the following appendices we perform data locality analysis of level set FD schemes using the model of Wolf and Lam [134]. For the sake of completeness, we briefly introduce the model and explain how to use it for analysis.

Considering a perfectly nested loop of depth n, we look at the iteration space which corresponds to a convex polyhedron in $\mathbb{Z}^n$ bounded by the loop bounds. We can identify each iteration by a node inside this polyhedron using a vector $\vec{p} = (p_1, p_2, \cdots, p_n)$, where p_i is the loop index of the i'th loop in the nest. An execution of the loop-nest

corresponds to visiting all nodes in the polyhedron in lexicographic order. We have reuse of a data item if it is accessed in several iterations of the loop. Thus, reuse is inherent in a computation and does not depend on the execution order of the loops in the nest. However, reuse does not guarantee temporal or spatial memory locality since accesses to a particular data item might be separated by many accesses to other data. This means that in the worst case the data item will have to be loaded each time it is used. The space spanned by the iteration space directions in which reuse is found is called the *reuse vector space*. We can transform our iteration space which corresponds to performing loop transformations such as skewing. These transformations change the way the iteration space is traversed and thus the way we exploit reuse. We must however be sure that the data dependencies of the algorithm are not violated. The dependence vectors, which define dependencies between two nodes $\vec{p}_1$ and $\vec{p}_2$ in the iteration space, must be transformed as well. A dependence vector points from $\vec{p}_1$ to $\vec{p}_2$ if the execution of the statement at $\vec{p}_2$ depends on the result from $\vec{p}_1$. Hence a valid code transformation T must satisfy $T(\vec{p}_1) < T(\vec{p}_2)$, where $<$ is the lexicographic ordering.

While transformations might improve our utilization of reuse, they cannot alone exploit reuse in multiple dimensions. Therefore we also need to perform tiling. We can tile loops i through j (for $i < j$) if they are *fully permutable*, *i.e.* can be interchanged freely: A property satisfied if the dependence vectors are non-negative and have either lexicographically positive components d_1 through d_{i-1}, or components d_i through d_j which are non-negative. Thus we can also use transformations to enable tiling since the dependence vectors change.

The result of applying the transformations is a vector space spanned by the iteration directions in which reuse can be exploited. This vector space is called the *localized vector space*. The goal of our data locality analysis thus is: Given an iteration space and corresponding data dependence vectors, we want to apply skewing and tiling transformations in order to obtain a localized vector space which completely contains the reuse vector space. By ensuring this, data reuse will result in data locality allowing data to be accessed multiple times whilst in memory, thus avoiding data to be loaded from disk multiple times.

A.1 Forward Euler

Algorithm 1 Euler with first order upwind (original)

```
1: for t ← 0, T do
2:     for x ← 0, X do
3:         A[t+1,x] ← step(A[t,x],A[t,x−1],A[t,x+1])
4:     end for
5: end for
```

In this appendix we analyze the *forward Euler* algorithm in the model of Wolf and Lam [134]. In particular we derive the reuse vector space, propose specific loop transformations and then show that these transformations result in a localized vector space that includes the reuse vector space, hence exploiting maximal reuse. For the

sake of simplicity, we carry out the analysis in one spatial dimension and explain how this analysis generalizes to higher spatial dimensions.

Consider the pseudo-code for forward Euler in algorithm 1. To facilitate the analysis, we assume the existence of an array A that includes both the spatial and temporal dimensions. Furthermore we assume that the initial data is present in $A[0,x]$ for all x in the spatial dimension, and that the upper loop bounds are not included in the iteration space (*i.e.* zero-based indexing). Note that in the actual implementation, our algorithms retain the iteration order illustrated in algorithm 1, but only a thin narrow band is traversed, and iterators into the DT-Grid data structure are used to abstract away the details of topology encoding and streaming to and from disk. Furthermore we use the storage allocation scheme described in section 3.1.

As explained in the previous appendix, we need to adhere to the dependencies of the algorithm when we transform the iteration space, *i.e.* we must ensure that an iteration does not execute before an iteration which it depends on. The dependence vectors are $\{(1,0),(1,-1),(1,1)\}$. To find the reuse vector space, we classify the memory references in terms of how they reuse memory. To facilitate this, we put two references $A[f(\vec{i})]$ and $A[g(\vec{i})]$ in the same equivalence class, called a *uniformly generated set*, if it for some linear transformation H and constant vectors $\vec{c_f}$ and $\vec{c_g}$ holds that $f(\vec{i}) = H\vec{i} + \vec{c_f}$ and $g(\vec{i}) = H\vec{i} + \vec{c_g}$.

All references in algorithm 1 are uniformly generated with $H = \mathrm{Id}$ and constant vectors

$$c_1 = (1,0)^T, c_2 = (0,0)^T, c_3 = (0,-1)^T, c_4 = (0,1)^T,$$

respectively. We wish to compute the reuse vector space containing all the directions of reuse, and therefore we look at group-spatial reuse (*i.e.* reuse as a result of references that are close in either time or space), as the associated reuse space R_{GS} contains all the other types of reuse. R_{GS} is defined as $\mathrm{span}\{r_2,\ldots,r_4\} + \ker H_S$, where H_S denotes H with the last row replaced by the zero-vector, and r_j for $j = 2,\ldots,4$ is a particular solution of the linear equation $H\vec{r}_j = \vec{c_{S,1}} - \vec{c_{S,j}}$ where $\vec{c_{S,i}}$ is $\vec{c}_i$ with the last component set to 0. Since $\ker H_S = \mathrm{span}\{(0,1)\}$, and the particular solutions all lie in $\mathrm{span}\{(1,0)\}$, we get that $R_{GS} = \mathrm{span}\{(1,0),(0,1)\}$. We can thus see that there is reuse in the entire iteration space. The localized vector space for the original loops is however only $L = \mathrm{span}\{(0,1)\}$. Hence R_{GS} is not fully contained in L, and in particular we only exploit reuse in the innermost loop. To improve this, we want to tile our loop nest in all necessary directions. However, it is not possible to do tiling in both loops as they are not fully permutable. Therefore, we have to skew our loops using the following transformation $T : [t,x] \rightarrow [t,x+t]$. The result is shown in algorithm 2. This also transforms our dependence vectors: $\{(1,1),(1,0),(1,2)\}$, and we can now clearly see, that our loops are fully permutable. Tiling the t and x loops with tile sizes B_T and B_X respectively, reveals the pseudocode in algorithm 3 for which the localized vector space completely coincides with R_{GS} thus demonstrating that our skewing and tiling scheme exploits all reuses.

To minimize the IO latency and bandwidth of our computations in practice, we do not tile in the x-direction, independent of the number of spatial dimensions involved in the computations. Note that by not tiling in the x-direction we do not alter the

Algorithm 2 Euler with first order upwind (skewed)

1: **for** $t \leftarrow 0, T$ **do**
2: **for** $x \leftarrow t, X+t$ **do**
3: $A[t+1,x-t] \leftarrow \text{step}(A[t,x-t],A[t,x-1-t],A[t,x+1-t])$
4: **end for**
5: **end for**

Algorithm 3 Euler with first order upwind (skewed and tiled)

1: **for** $t_2 \leftarrow 0, T, B_T$ **do**
2: **for** $x_2 \leftarrow 0, X+T, B_X$ **do**
3: **for** $t \leftarrow t_2, \min(T, t_2+B_T)$ **do**
4: **for** $x \leftarrow \max(t,x_2), \min(X+t, x_2+B_X)$ **do**
5: $A[t+1,x-t] \leftarrow \text{step}(A[t,x-t],A[t,x-1-t],A[t,x+1-t])$
6: **end for**
7: **end for**
8: **end for**
9: **end for**

localized vector space. This is due to the fact that the loop within the tile in the x-direction can become the outermost loop - among those loops that iterate *within* a tile - by simple loop-interchange. Furthermore, the innermost controlling loop can trivially be coalesced with the loop over its individual tiles since all loops are fully permutable. Algorithm 4 shows our final algorithm for forward Euler.

Generalization to higher order spatial schemes and higher spatial dimensions is straightforward. To use the *Hamilton-Jacobi Weighted ENO* (HJ WENO) [51, 52, 69] scheme, the transformation would skew by three instead of one, and thus becomes $T : [t,x] \rightarrow [t,x+3t]$. In higher spatial dimensions, we skew identically and tile in each additional spatial dimension.

Algorithm 4 Euler with first order upwind (final)

1: **for** $t_2 \leftarrow 0, T, B_T$ **do**
2: **for** $x \leftarrow 0, X+T$ **do**
3: **for** $t \leftarrow t_2, \min(T, t_2+B_T)$ **do**
4: **if** $\max(t,x) < \min(X+t, x+1)$ **then**
5: $A[t+1,x-t] \leftarrow \text{step}(A[t,x-t],A[t,x-1-t],A[t,x+1-t])$
6: **end if**
7: **end for**
8: **end for**
9: **end for**

A.2 BFECC AND TVD RK

We now proceed to analyze the *Back and Forth Error Compensation and Correction* (BFECC) [21] and *Total Variation Diminishing Runge Kutta* (TVD RK) [112] algo-

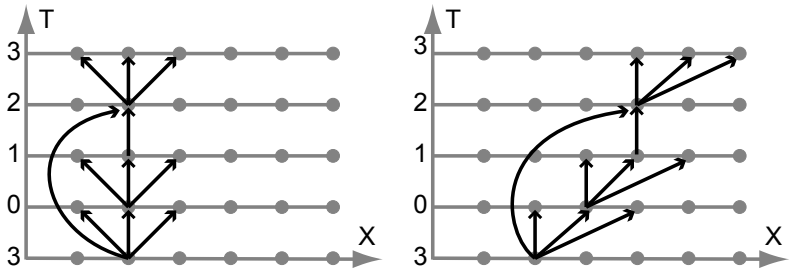


Figure 22 Dependence vectors for the BFECC algorithm with first order spatial derivatives. The numering on the T axis is $t$mod4. **Left**: Before skewing transformation. **Right:** After skewing transformation.

Algorithm 5 BFECC with first order upwind (original)

```
1:  for t ← 0, 4T do
2:      for x ← 0, X do
3:          if t mod 4 = 0 then
4:              A[t+1,x] ← step(A[t,x−1],A[t,x],A[t,x+1])
5:          else if t mod 4 = 1 then
6:              A[t+1,x] ← backstep(A[t,x−1],A[t,x],A[t,x+1])
7:          else if t mod 4 = 2 then
8:              A[t+1,x] ← average(A[t,x],A[t−2,x])
9:          else
10:             A[t+1,x] ← step(A[t,x−1],A[t,x],A[t,x+1])
11:         end if
12:     end for
13: end for
```

rithms. The procedure of the analysis is similar for the two algorithms as they both consist of a number of propagation and weighted averaging steps. Due to the relative simplicity of the BFECC algorithm we will focus on this algorithm and only provide an outline of the TVD RK analysis along with the proposed transformations. Again, we carry out the analysis in one spatial dimension and then generalize it to higher spatial dimensions.

Consider the pseudo-code for the BFECC algorithm shown in algorithm 5. To facilitate the analysis we have formulated the BFECC algorithm as a perfect loop nest by introducing the fictitious time variable t such that $\lfloor \frac{t}{4} \rfloor$ denotes the time-step and t mod 4 uniquely identifies one of the four assignment statements in the loop body. Furthermore, we have again assumed the existence of an array A that includes both the spatial and temporal dimension, and holds the initial data in $A[0,x]$ for all x in the spatial dimension. The storage mapping scheme can be found in section 3.1.

Assuming the j'th array reference is expressed as $H\,[t,x]^T + \vec{c}_j$, only a single uniformly generated set of references is present as represented by the identity matrix $H = \mathrm{Id}$. Since $\ker H_S = \mathrm{span}\{(0,1)\}$, and $\mathrm{span}\{\vec{r}_j\} = \mathrm{span}\{(1,0)\}$, we conclude that $R_{\mathrm{GS}} = \mathrm{span}\{(1,0),(0,1)\}$, hence the BFECC algorithm has reuse in both the temporal

Algorithm 6 BFECC with first order upwind (skewed)

```
1:  for t ← 0, 4T do
2:      x_start ← 3⌊t/4⌋ + min(t mod 4, 1) + ⌊(t mod 4)/3⌋
3:      for x ← x_start, X + x_start do
4:          w ← x − x_start
5:          if t mod 4 = 0 then
6:              A[t+1,w] ← step(A[t,w−1],A[t,w],A[t,w+1])
7:          else if t mod 4 = 1 then
8:              A[t+1,w] ← backstep(A[t,w−1],A[t,w],A[t,w+1])
9:          else if t mod 4 = 2 then
10:             A[t+1,w] ← average(A[t,w],A[t−2,w])
11:         else
12:             A[t+1,w] ← step(A[t,w−1],A[t,w],A[t,w+1])
13:         end if
14:     end for
15: end for
```

and spatial dimensions.

To fully exploit reuse, we must find loop transformations that result in a localized vector space L, such that $R_{GS} \subset L$ without violating the dependencies of the BFECC algorithm. Note that since the loop body has four separate cases, $t \bmod 4 = \{0,1,2,3\}$, we must consider the dependence vectors of *each* of these cases locally. For cases 0 and 2 they are $\{(1,-1),(1,0),(1,1)\}$, for case 1 the dependence vector is $\{(1,0)\}$ and for case 3 the dependence vectors are $\{(1,-1),(1,0),(1,1),(3,0)\}$ as depicted to the left in figure 22. Recall that the loops are fully permutable, and hence tilable, if all entries in the transformed dependence vectors are non-negative. Thus we see from the dependence vectors that by skewing by one in the x-direction from one case to the next, except from case 1 to case 2, we obtain a fully permutable loop nest. Specifically, we propose the skewing transformation $T : [t,x] \rightarrow [t, x+3\lfloor \frac{t}{4} \rfloor + \min(t \bmod 4, 1) + \lfloor \frac{t \bmod 4}{3} \rfloor]$ which results in dependence vectors, (t,x), equal to $\{(1,0),(1,1),(1,2),(3,2)\}$ for case 3, $\{(1,0),(1,1),(1,2)\}$ for cases 0 and 2, and $\{(1,0)\}$ for case 1 as depicted to the right in figure 22. Note that the proposed transformation is not constant throughout the iteration space. However the proposed transformation is invariant in the x-direction for each case, since $T((t_1,x_1)) - T((t_2,x_2)) = T((t_1,x_1+d)) - T((t_2,x_2+d))$, so we can transform the end-points of the dependence vectors independent of their absolute position in the x-direction.

Algorithm 6 shows the code resulting from skewing the iteration space. In particular the loop bounds are transformed using T and the array indices are transformed using T^{-1}. To obtain an optimal localized vector space, tiling and loop interchange is performed in algorithm 7. To do the actual loop interchange in algorithm 7 between the controlling loop in the x-direction and the loop within the tile for the temporal dimension we do the following: The bounds of the controlling loop in the x-direction are made independent of t by exchanging the minimum value for t in the lower bound and the maximum value for t in the upper bound. When doing this it is important to change the lower bound of the loop within the tile in the x-direction to $\max(x_2, x_{\text{start}})$, since

Algorithm 7 BFECC with first order upwind (skewed and tiled)

```
1:  for t_2 ← 0, 4T, B_t do
2:      for x_2 ← 0, X + 3⌊(4T−1)/4⌋ + 2, B_x do
3:          for t ← t_2, min(4T, t_2 + B_t) do
4:              x_start ← 3⌊t/4⌋ + min(t mod 4, 1) + ⌊(t mod 4)/3⌋
5:              for x ← max(x_2, x_start), min(x_2 + B_x, X + x_start) do
6:                  w ← x − x_start
7:                  if t mod 4 = 0 then
8:                      A[t+1, w] ← step(A[t, w−1], A[t, w], A[t, w+1])
9:                  else if t mod 4 = 1 then
10:                     A[t+1, w] ← backstep(A[t, w−1], A[t, w], A[t, w+1])
11:                 else if t mod 4 = 2 then
12:                     A[t+1, w] ← average(A[t, w], A[t−2, w])
13:                 else
14:                     A[t+1, w] ← step(A[t, w−1], A[t, w], A[t, w+1])
15:                 end if
16:             end for
17:         end for
18:     end for
19: end for
```

this ensures that the loop within the tile starts at either the lower bound of the iteration space or at the lower bound of a tile within the iteration space. As in the case of the forward Euler algorithm, we do not tile in the x-direction, and the localized iteration space remains unchanged. In algorithm 8 we show the BFECC algorithm that results from only tiling in the temporal dimension.

Generalization to higher order spatial schemes and higher spatial dimensions is again straightforward. To use the HJ WENO scheme, the transformation would skew by three instead of one in each case (except case 3 in which there is still no skewing), and thus becomes $T : [t,x] \rightarrow [t, x + 9\lfloor \frac{t}{4} \rfloor + 3\min(t \bmod 4, 1) + 3\lfloor \frac{t \bmod 4}{3} \rfloor]$. In higher spatial dimensions, we skew identically and tile in each additional spatial dimension, in two dimensions for example we obtain $T : [t,x,y] \rightarrow [t, x + 3\lfloor \frac{t}{4} \rfloor + \min(t \bmod 4, 1) + \lfloor \frac{t \bmod 4}{3} \rfloor, y + 3\lfloor \frac{t}{4} \rfloor + \min(t \bmod 4, 1) + \lfloor \frac{t \bmod 4}{3} \rfloor]$.

For a TVD RK method the analysis proceeds analogously. Take for example the third order accurate TVD RK scheme which consists of three advection steps and two averaging steps, as opposed to three advection steps and one averaging step in the case of the BFECC scheme. In particular the fictitious time includes a multiplicative factor of 5 (instead of 4 for BFECC), and the transformation for 1D TVD RK combined with a first order upwind scheme in the spatial dimension becomes $T : [t,x] \rightarrow [t, x + 3\lfloor \frac{t}{5} \rfloor + \min(t \bmod 5, 1) + \lfloor \frac{t \bmod 5}{3} \rfloor]$. This can readily be seen from a diagram similar to the one shown in figure 22. The generalization to higher spatial dimensions and higher order spatial schemes is identical to that of the BFECC scheme. In particular skewing is applied in each spatial dimension and the magnitude of skewing increases respectively.

Algorithm 8 BFECC with first order upwind (final)

for $t_2 \leftarrow 0, 4T, B_t$ **do**
  for $x \leftarrow 0, X + 3\lfloor \frac{4T-1}{4} \rfloor + 2$ **do**
    for $t \leftarrow t_2, \min(4T, t_2 + B_t)$ **do**
      $x_{\text{start}} \leftarrow 3\lfloor \frac{t}{4} \rfloor + \min(t \bmod 4, 1) + \lfloor \frac{t \bmod 4}{3} \rfloor$
      if $\max(x, x_{\text{start}}) < \min(x+1, X + x_{\text{start}})$ **then**
        $w \leftarrow \max(x, x_{\text{start}}) - x_{\text{start}}$
        if $t \bmod 4 = 0$ **then**
          $A[t+1, w] \leftarrow \text{step}(A[t, w-1], A[t, w], A[t, w+1])$
        else if $t \bmod 4 = 1$ **then**
          $A[t+1, w] \leftarrow \text{backstep}(A[t, w-1], A[t, w], A[t, w+1])$
        else if $t \bmod 4 = 2$ **then**
          $A[t+1, w] \leftarrow \text{average}(A[t, w], A[t-2, w])$
        else
          $A[t+1, w] \leftarrow \text{step}(A[t, w-1], A[t, w], A[t, w+1])$
        end if
      end if
    end for
  end for
end for

PAPER II

GUIDING OF SMOKE ANIMATIONS THROUGH VARIATIONAL COUPLING OF SIMULATIONS AT DIFFERENT RESOLUTIONS

Michael Bang Nielsen Brian Bunch Christensen Nafees Bin Zafar
Doug Roble Ken Museth

Abstract

We propose a novel approach to guiding of Eulerian-based smoke animations through coupling of simulations at different grid resolutions. Specifically we present a variational formulation that allows smoke animations to adopt the low-frequency features from a lower resolution simulation (or non-physical synthesis), while simultaneously developing higher frequencies. The overall motivation for this work is to address the fact that art-direction of smoke animations is notoriously tedious. Particularly a change in grid resolution can result in dramatic changes in the behavior of smoke animations, and existing methods for guiding either significantly lack high frequency detail or may result in undesired features developing over time. Provided that the bulk movement can be represented satisfactorily at low resolution, our technique effectively allows artists to prototype simulations at low resolution (where computations are fast) and subsequently add extra details without altering the overall "look and feel". Our implementation is based on a customized multi-grid solver with memory-efficient data structures.

Published as: Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble and Ken Museth. Guiding of Smoke Animations Through Variational Coupling of Simulations at Different Resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (New Orleans, LA, USA, August 1–2, 2009). Symposium on Computer Animation 2009. ACM, New York, NY, USA, 217–226.

Figure 1 A velocity, density, and temperature source injects hot high-velocity smoke subject to a buoyancy force. **Left:** Unguided simulation (256^3). **Middle:** Unguided simulation (64^3). **Right:** 256^3 simulation guided by the 64^3 simulation. There is poor resemblance between the unguided simulations. The guided simulation follows the general flow of the low resolution simulation, and adds dynamic high frequency detail.

1 Introduction

Grid-based Navier-Stokes solvers, like [26, 114], are commonly used for smoke effects in movie production, but they require significant CPU and memory resources. This is especially problematic in the context of visual effects since photorealistic smoke typically call for simulations on high-resolution grids. Simulation effects often require many iterations to create the desired look. In order to have quick turn-around times, artists often perform their art-direction iterations at a low resolution to determine the most effective simulation setup, and then run the simulations at final high resolutions. However, the change of resolution often completely changes the overall "look" of the animation, which may cause the composition to fail from the director's viewpoint [36]. This property is especially evident when the initial grid resolution is very low and the solution has far from converged. It is primarily caused by the numerical viscosity of the discretization and is in turn further enhanced by the inherent non-linearity of the Navier-Stokes equations.

There is no set methodology for scaling simulation settings such that the high resolution simulation matches the look. Instead the artist has to intuit the alterations from experience and engage in an iterative process in high resolution as well.

The goal of this paper is to propose a technique for Eulerian grid-based smoke simulations [26, 114] that allows animators to use low-resolution input simulations to *guide* higher-resolution ones in such a way that details are added, but the overall (*i.e.* low frequency) flow is preserved as well as possible. Therein lies the assumption that the desired overall bulk movement is representable by the low resolution simulation , and our approach presupposes that this is the case. Our method is comprised of a physically based fluid simulation combined with a novel pressure projection step that minimizes the deviation of the velocity field's low frequencies from the low-resolution guiding flow, whilst constraining the velocity field to be divergence free.

The feasibility of guiding, or tracking, based on low resolution input simulations has previously been demonstrated by Bergou *et al.* [7] for thin shells and by Thürey *et al.* [123] for Lattice Boltzmann (LBM) and Smoothed Particle Hydrodynamics (SPH) simulations of liquids. Like Bergou *et al.* we analyze the problem in a mathematical framework and formulate guiding as a set of constrained equations. The motivation

that led us to take this approach was that simpler strategies, considered to some extent by previous work, did not suffice. In particular upsampling a low-resolution velocity field followed by pressure projection (see Figure 4.c) or blending the velocity field with a guiding velocity field (see discussion in [123]) is incapable of producing higher frequencies and causes significant smoothing of high frequency detail, respectively. Another relatively simple strategy is to blend the low frequencies of the velocity field with a low-resolution guiding simulation before pressure projection. This is essentially the idea presented by Thürey *et al.* except that they consider guiding particles. However this approach may, in our experience (see section 8), introduce undesired features over time when applied to an Eulerian-based smoke simulation. We hypothesize that this is due in part to the loss of explicit control over the low frequencies during the standard pressure projection step.

Our proposed workflow consists of first applying existing fluid control methodologies to create a low resolution simulation. Next step is to invoke our framework with the low resolution simulation as input, possibly iteratively to control several frequency bands separately. Additional sub-grid motion details can then be generated using one of the recently developed turbulence synthesis methods [58, 86, 107].

Provided that an appropriate low-resolution simulation can be found, our framework allows the look development of the bulk movement to take place in low resolution. Thereby it offers a starting point in high resolution for control that pertains directly to high frequency features. In particular this paper claims the following novel contributions over previous work:

- A variational formulation of a guiding velocity field and a resulting set of linear equations.
- A practical implementation of our guiding framework, including methods for lowpass filter estimation and handling of boundaries.
- A custom multi-threaded multigrid implementation based on a fast and compact dynamic matrix storage format.

2 Related Work

The concept of high level animation control of full Navier-Stokes based simulations was first advocated by Foster and Metaxas[32]. Later Treuille *et al.* [126] proposed a gradient descent based optimization framework for keyframe control of smoke simulations. Specifically the framework optimizes for forces that result in the fluid assuming keyframed poses. McNamara *et al.* [75] improved the speed of this framework by several orders of magnitude by adopting the adjoint method. While powerful and generic, simulation time seems to limit the resolutions feasible with this approach, as an entire fluid simulation has to be run for each step in the optimization.

Thürey *et al.* [123] successfully demonstrated guiding simulations for SPH and LBM. In particular, they propose a fast technique for controlling low frequencies of

liquid animations by applying a combination of feedback and attraction forces. Contrary to our work, [123] is force-based, which (as noted in their paper) makes it difficult to enforce constraints on the velocity field during pressure projection. Thürey *et al.* do not directly address this problem, but refer to the work of Shi *et al.* [110] for a possible solution. Shi *et al.* consider the problem of making liquid simulations follow rapidly changing target animations. They propose a force-based solution that is comprised of a velocity- and shape-feedback force as well as a potential function. The shape-feedback force of Shi *et al.* is designed to be divergence free, and is therefore unaffected by the pressure projection step. However, the shape-feedback force is specifically designed for target shapes and is not well suited for smoke simulations with no specific target shapes.

Recent work has also successfully focused on improving run times and memory usage of full Navier-Stokes simulations by developing procedural methods that synthesize the high frequency detail, given a low resolution simulation as input [58, 86, 107]. Although the goal of our work bears similarities with these synthesis methods we emphasize that it is fundamentally different in that we partly simulate the high frequency detail. Our method also allows for non-physically based inputs which enhances the ability of animators to art-direct fluid animations. We stress that our work in no way precludes the procedural synthesis methods, but is meant to work with such techniques to produce high resolution simulations quickly.

Concurrently with our work, Mullen *et al.* [79] developed energy preserving integrators for simplicial grids. These integrators can also be applied to obtain higher resemblance between low and high resolution simulations.

3 Algorithm Overview

The flow of an inviscid, incompressible fluid is described by the inviscid Euler equations $\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\,\mathbf{v} = -\nabla p + \mathbf{f}$ and $\nabla \cdot \mathbf{v} = 0$, where $\mathbf{v}$ denotes the velocity of the fluid, p denotes pressure, $\mathbf{f}$ is an external force, and the fluid density is assumed to be 1 for simplicity. In computer graphics, this set of equations is often solved using the operator splitting approximation described in [114]. The central idea is to decouple self-advection, addition of external forces and enforcement of incompressibility by sequentially solving for each of these terms. As illustrated in Figure 2, our method only requires modification of the pressure projection step that solves for the latter term. Specifically, our method proceeds as following for each iteration of the guided, high-resolution simulation:

1. Obtain the low-resolution, guiding velocity field $\mathbf{v}_{\text{low}}$ through simulation or art direction.
2. Upsample $\mathbf{v}_{\text{low}}$ to high resolution.
3. Advect and add forces using traditional methods to obtain the intermediate velocity field $\tilde{\mathbf{v}}$ in high resolution.
4. Obtain the guiding weights α through art direction or an automated process.

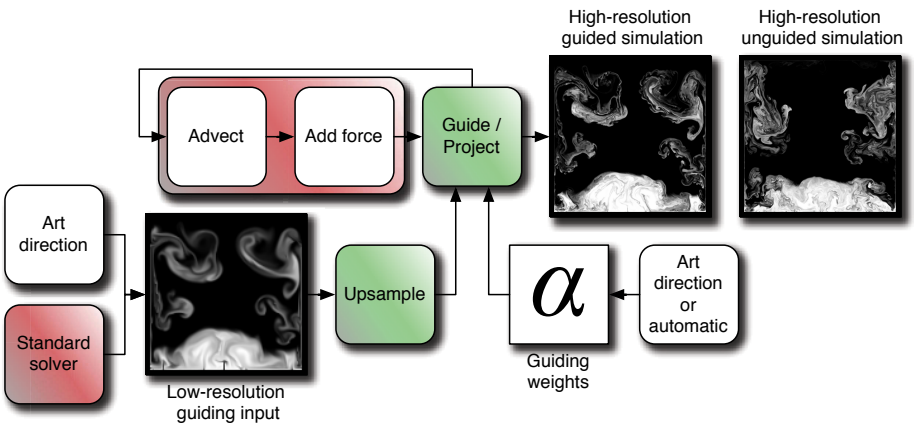


Figure 2 Our method consists of two components (in green) which are easily integrated in a traditional fluid simulation pipeline (in red). The low-resolution input velocity field is upsampled for the guided projection step, which ensures a similar bulk movement of the high-resolution simulation but with added detail. An unguided simulation of high resolution is provided for reference.

5. Solve a modified projection step using the upsampled $\mathbf{v}_{\text{low}}$, the guiding weights α and $\tilde{\mathbf{v}}$ to obtain the new high-resolution, guided velocity field $\mathbf{v}$.

The guiding weights α will be explained in the following section.

4 Variational Model of Guiding

In this section we analyze guiding velocity fields for fluid animation in a mathematical framework. As discussed in the introduction, existing simpler methods do in our experience not suffice. Motivated by this, we consider guiding of fluids as a constrained variational problem. Furthermore we derive the resulting linear equations that solve for the stationary point of this variational problem. We first derive the variational problem continuously but could equally well have derived the linear equations directly from a specific discretization. As a prelude we briefly summarize how the pressure projection step can be regarded as a variational problem.

4.1 Preliminaries

The Poisson equation

$$\nabla \cdot \tilde{\mathbf{v}} = \Delta p \tag{II.1}$$

combined with $\mathbf{v} = \tilde{\mathbf{v}} - \nabla p$ enforces the continuity condition of the inviscid Euler equations, where $\mathbf{v}$ and $\tilde{\mathbf{v}}$ are velocity fields, $\mathbf{v}$ is divergence free and p is pressure. It can be derived from the minimization of the difference between two velocity fields $\mathbf{v}$ and $\tilde{\mathbf{v}}$ subject to the continuity constraint that $\mathbf{v}$ be divergence free (see [29] pp.

202-204). Mathematically this amounts to minimizing

$$\frac{1}{2}\int_{\Omega}|\mathbf{v}(\mathbf{r})-\tilde{\mathbf{v}}(\mathbf{r})|^2\,d\mathbf{r} \tag{II.2}$$

subject to the constraint

$$\nabla\cdot\mathbf{v}(\mathbf{r})=0 \tag{II.3}$$

where $\mathbf{r}$ is the position vector and Ω is the fluid domain. Eq. (II.2) and Eq. (II.3) can be combined into the saddle point problem

$$\int_{\Omega}\left\{\frac{1}{2}|\mathbf{v}(\mathbf{r})-\tilde{\mathbf{v}}(\mathbf{r})|^2-\lambda(\mathbf{r})\nabla\cdot\mathbf{v}(\mathbf{r})\right\}d\mathbf{r} \tag{II.4}$$

where $\lambda(\mathbf{r})$ are scalar-valued Lagrange multipliers. Taking the first variation and solving for a stationary point, one obtains Eq. (II.1) and Eq. (II.3) with λ replacing p (hence "'pressure"' is, in fact, a Lagrange multiplier).

4.2 Guiding Equations

To let a given low-resolution velocity field, $\mathbf{v}_{\text{low}}$, guide a high-resolution velocity field, $\mathbf{v}$, we add an additional term to Eq. (II.2) and Eq. (II.3). The new term prescribes that a *smoothed version of the high-resolution velocity field should be as close as possible to the low-resolution input velocity field upsampled to high resolution*. Enforcing that the convolution of the high-resolution velocity field should be *identical* (by imposing a constraint) to the low-resolution input velocity field upsampled to high resolution leads to ill-posed problems for most lowpass filters (*i.e.* a (unique) solution does not exist). Mathematically, our term is formulated as

$$R=\frac{1}{2}\int_{\Omega}|[\mathscr{F}*\mathbf{v}](\mathbf{r})-\mathbf{v}_{\text{low}}(\mathbf{r})|^2\,d\mathbf{r} \tag{II.5}$$

where $\mathscr{F}$ is a, possibly spatially varying, lowpass filter, $*$ denotes a convolution and $\mathbf{v}_{\text{low}}$ is the low-resolution input velocity field upsampled to high resolution. Next we combine Eq. (II.4) and Eq. (II.5) into the saddle point problem

$$\begin{aligned}\int_{\Omega}\Bigg\{&\frac{\alpha(\mathbf{r})}{2}|\mathbf{v}(\mathbf{r})-\tilde{\mathbf{v}}(\mathbf{r})|^2-\lambda(\mathbf{r})\nabla\cdot\mathbf{v}(\mathbf{r})\\&+\frac{(1-\alpha(\mathbf{r}))}{2}|[\mathscr{F}*\mathbf{v}](\mathbf{r})-\mathbf{v}_{\text{low}}(\mathbf{r})|^2\Bigg\}d\mathbf{r}\end{aligned} \tag{II.6}$$

where $\alpha\in(0;1]$ is a scaling parameter that determines the relative weight of each of the terms. In section 8 we will show how α can be used to introduce artistic control. For now we make the following observations; 1) α can vary both spatially and temporally, 2) for $\alpha=1$ we obtain a normal unguided fluid simulation and 3) the solution to the saddle point problem, $\mathbf{v}$, is divergence free due to the constraint enforced by the Lagrange multipliers. To solve this saddle point problem for $\mathbf{v}$ we employ the calculus of variations. This amounts to deriving the stationary points of Eq. (II.6) that have zero first variation. Here we focus on the first variation, δR, of our term R given by

Eq. (II.5), and next combine with the first variation of Eq. (II.4) considered in [29]. Assume in the following that the velocity field $\mathbf{v}$ gives rise to the desired stationary point, $R_{\text{stationary}}$. That is, $R_{\text{stationary}} = R(\mathbf{v})$. Next we consider $\delta\mathbf{v}$ to be the variation of $\mathbf{v}$, and ignore second order terms (*i.e.* terms of the kind $\delta\mathbf{v}^2$) that do not contribute to the first variation. The first variation, δR, then reads as

$$\begin{aligned}\delta R &= R - R_{\text{stationary}} = R(\mathbf{v}+\delta\mathbf{v}) - R(\mathbf{v}) \\ &= \int_\Omega \left[\mathscr{F} * \delta\mathbf{v}\right](\mathbf{r}) \cdot \left[\left[\mathscr{F} * \mathbf{v}\right](\mathbf{r}) - \mathbf{v}_{\text{low}}(\mathbf{r})\right] d\mathbf{r} \\ &= \int_\Omega \delta\mathbf{v}(\mathbf{r}) \cdot \int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r}) \left[\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) - \mathbf{v}_{\text{low}}(\mathbf{q})\right] d\mathbf{q} d\mathbf{r}\end{aligned}$$

which holds from the definition of convolution, the linearity of the integral, and the definition and distributivity of the dot product. The integration variable $\mathbf{q}$ arises from the expansion of $\mathscr{F} * \delta\mathbf{v}$ into its integral definition. Since $\delta\mathbf{v}(\mathbf{r})$ is arbitrary in all points of the domain, and $\delta R = 0$ for a stationary point, we obtain the following necessary and sufficient condition for a stationary point of Eq. (II.5):

$$\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r}) \left[\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) - \mathbf{v}_{\text{low}}(\mathbf{q})\right] d\mathbf{q} = 0$$

Combining this integral with the integral obtained for the first variation of Eq. (II.4), derived in [29], and including the scaling parameter $\alpha > 0$, a necessary and sufficient condition for a stationary point of Eq. (II.6) is:

$$\begin{aligned}&\mathbf{v}(\mathbf{r}) + \tfrac{1}{\alpha(\mathbf{r})}\nabla\lambda(\mathbf{r}) + \tfrac{1}{\alpha(\mathbf{r})}\textstyle\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) d\mathbf{q} \\ &\quad = \tilde{\mathbf{v}}(\mathbf{r}) + \tfrac{1}{\alpha(\mathbf{r})}\textstyle\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q}) d\mathbf{q}\end{aligned} \tag{II.7}$$

where the right hand side is known (the reason for dividing by α is that the implementation of Eq. (II.8) simplifies). The linear system obtained by discretizing and combining Eq. (II.7) with the constraints in Eq. (II.3) is however not well suited for relaxation methods such as Gauss Seidel which forms part of a multigrid implementation. This is due to the fact that Gauss Seidel employs a division by the diagonal entry, which is zero in the lower part of the matrix corresponding to Eq. (II.3) since no terms including λ are present. In order to obtain a linear system with non-zero diagonal entries in all rows, we apply the constraint Eq. (II.3) to Eq. (II.7) instead of discretizing Eq. (II.3) explicitly:

$$\begin{aligned}&\nabla\cdot\left(\tfrac{1}{\alpha(\mathbf{r})}\nabla\lambda(\mathbf{r}) + \tfrac{1}{\alpha(\mathbf{r})}\textstyle\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) d\mathbf{q}\right) \\ &\quad = \nabla\cdot\left(\tilde{\mathbf{v}}(\mathbf{r}) + \tfrac{1}{\alpha(\mathbf{r})}\textstyle\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q}) d\mathbf{q}\right)\end{aligned} \tag{II.8}$$

where we have used that $\nabla\cdot\mathbf{v}(\mathbf{r}) = 0$. If α is not spatially varying, Eq. (II.7) becomes

$$\begin{aligned}&\mathbf{v}(\mathbf{r}) + \tfrac{1}{\alpha}\nabla\lambda(\mathbf{r}) + \tfrac{(1-\alpha)}{\alpha}\textstyle\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) d\mathbf{q} \\ &\quad = \tilde{\mathbf{v}}(\mathbf{r}) + \tfrac{(1-\alpha)}{\alpha}\textstyle\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q}) d\mathbf{q}\end{aligned} \tag{II.9}$$

and Eq. (II.8) simplifies to

$$\begin{aligned}&\tfrac{1}{\alpha}\Delta\lambda(\mathbf{r}) + \tfrac{(1-\alpha)}{\alpha}\nabla\cdot\textstyle\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q}) d\mathbf{q} \\ &\quad = \nabla\cdot\tilde{\mathbf{v}}(\mathbf{r}) + \tfrac{(1-\alpha)}{\alpha}\nabla\cdot\textstyle\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q}) d\mathbf{q}\end{aligned} \tag{II.10}$$

$$\left(\begin{array}{c|c} \vdots & \vdots \\ \mathbf{v}(\mathbf{r}) + \frac{(1-\alpha)}{\alpha}\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F}*\mathbf{v}\right](\mathbf{q})d\mathbf{q} & \frac{1}{\alpha}\nabla\lambda(\mathbf{r}) \\ \vdots & \vdots \\ \hline \vdots & \vdots \\ \frac{(1-\alpha)}{\alpha}\nabla\cdot\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F}*\mathbf{v}\right](\mathbf{q})d\mathbf{q} & \frac{1}{\alpha}\Delta\lambda(\mathbf{r}) \\ \vdots & \vdots \end{array}\right)\left(\begin{array}{c} \vdots \\ \mathbf{v}(\mathbf{r}) \\ \vdots \\ \hline \vdots \\ \lambda(\mathbf{r}) \\ \vdots \end{array}\right) = \left(\begin{array}{c} \vdots \\ \tilde{\mathbf{v}}(\mathbf{r}) + \frac{(1-\alpha)}{\alpha}\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q})d\mathbf{q} \\ \vdots \\ \hline \vdots \\ \nabla\cdot\tilde{\mathbf{v}}(\mathbf{r}) + \frac{(1-\alpha)}{\alpha}\nabla\cdot\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q})d\mathbf{q} \\ \vdots \end{array}\right)$$

Figure 3 Shows the linear system $Ax = b$ of $(D+1)N$ equations in $(D+1)N$ unknowns that results from combining Eq. (II.9) and Eq. (II.10) where D is the number of dimensions and N is the number of grid cells. Note that the matrix A acts as an operator on x. However to emphasize the relationship with Eq. (II.9) and Eq. (II.10) we have included the complete expressions in the matrix (this is why *e.g.* $\mathbf{v}(\mathbf{r})$ appears both as an unknown and in the matrix).

By combining and discretizing Eq. (II.7) and Eq. (II.8) (or Eq. (II.9) and Eq. (II.10) if α is not spatially varying), we get a linear system of $(D+1)N$ equations in $(D+1)N$ unknowns ($\mathbf{v}$ and λ), where D is the dimension and N is the number of grid points.

A few important properties of the derived equations are:

- The guiding velocity field, $\mathbf{v}_{\text{low}}$, does not have to be divergence free.
- The derived guiding equations are self-consistent in the sense that if a fluid velocity field, $\mathbf{v}$, is used to guide itself, meaning that if $\mathbf{v}_{\text{low}} = \mathscr{F} * \mathbf{v}$, then the result will be $\mathbf{v}$ itself. We have verified this experimentally.
- A solution to Eq. (II.7) and Eq. (II.8) exists simply because a divergence free velocity field exists. We leave a rigorous proof of the conditions required for uniqueness of the solution as future work. In practice it is our experience that the solution to the linear system converges to the desired precision in few multigrid cycles unless $\alpha \ll 1$.
- By *construction* of the variational problem, the low frequencies of the solution to the linear system will be a blend of the low frequencies in $\tilde{\mathbf{v}}$ and $\mathbf{v}_{\text{low}}$, and the contribution from each of these in the final velocity field can be controlled using α.

4.3 The Discretization of the Guiding Equations

We discretize the system of equations, Eq. (II.7-II.8) or Eq. (II.9-II.10), on a staggered MAC grid using finite difference approximations. That is, velocities are stored on cell faces, and the pressure/Lagrange multipliers are stored in cell centers. The operator matrix for the linear system, Eq. (II.9) combined with Eq. (II.10), is depicted in Figure 3. Note that we solve for all unknowns simultaneously. The gradient,$\nabla\lambda$, divergence, $\nabla\cdot\tilde{\mathbf{v}}$, and Laplacian, $\Delta\lambda$, are discretized using the usual second order approximations [26]. The term $\frac{(1-\alpha)}{\alpha}\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F}*\mathbf{v}\right](\mathbf{q})d\mathbf{q}$ is discretized by a sum of point-wise multiplications at cell faces ranging over the support of the lowpass filter. Let F be the discretized version of the lowpass filter, $\mathscr{F}$, then the discretization is

$\frac{(1-\alpha)}{\alpha}\sum_{\mathbf{q}\in\mathrm{supp}F} F(\mathbf{q}-\mathbf{r})\,[F*\mathbf{v}](\mathbf{q})$. Likewise the convolution and the right-hand-side correlation are implemented as a sum. The guiding terms involving the divergence are discretized by the usual second order approximation to the divergence. The upper part of the matrix, the first DN equations, are discretized on staggered cell faces, whereas the last N equations are discretized in cell centers. The resulting equation system is linear and sparse. However, the system of equations is asymmetric (see Figure 3) and due to the support of the lowpass filters, the matrix requires an impractical amount of memory when stored uncompressed. However, by employing the multigrid method [14], the system can be solved to sufficient precision in few multigrid cycles, and the matrix stored in a compact format.

5 Boundaries

A common complication in fluid simulations is boundary conditions. The situation is further complicated when dealing with simulations at multiple resolutions, since sampling issues might cause a voxelization of the boundary to differ between the simulations. We represent boundaries as the zero level set of a signed distance function sampled at the same resolutions as the simulations. This means that boundaries are limited to having features representable within the frequency space of the simulation resolutions to avoid sampling artifacts. Particularly, boundary features that are too thin to be represented properly in the low-resolution simulation might appear in the high-resolution simulation and potentially result in visual artifacts. We have left this issue as a direction for future work.

If the lowpass filter, $\mathscr{F}$, overlaps the boundaries, one might worry that visual artifacts could occur. We have explored relaxing the guiding weights near boundaries (in order to make the guiding more loose or disable it entirely). We have, however, not found this necessary, as we did not observe a qualitative improvement of the result. In fact, we observed that turning guiding off inside narrow pipes of fluid, caused the high-resolution fluid to move much faster than the guiding input due to less numerical dissipation, which is not desirable. We have also seen no major impact on convergence of the multigrid solver. However, if a large fraction of the domain is part of the boundary, disabling guiding in these parts of the domain speeds up the multigrid solver considerably.

5.1 The Penalization Method

Internal boundaries are traditionally hard to address in the multigrid paradigm. This is primarily due to the fact that the domain embedded by the internal boundaries is excluded by the multigrid solve, which in turn complicates the otherwise simple rectangular fluid domain. However, it is possible to treat internal boundaries implicitly, and basically solve for pressure and velocity everywhere in the computational domain, *i.e.* the multigrid solver itself is not explicitly aware of boundaries. One method that does exactly this is the *penalization method* of Angot *et al.* [4, 61]. This method also has the advantage of being non-iterative, which makes it a good candidate for

our application. An alternative iterative method for handling boundaries in multigrid context was recently proposed in [78]. The penalization method solves for pressure and velocity everywhere. However, inside boundaries, the velocity, $\mathbf{v}$, is penalized towards the prescribed velocity of the boundary, $\bar{\mathbf{v}}$, and at the surface of the boundary, the velocity satisfies the no-slip condition (the Euler equations, being inviscid, are subject to free-slip, so we rely here on numerical viscosity to make the problem well-posed). Mathematically the penalization is achieved by adding a penalization term, $\frac{1}{\eta}\chi_{\text{boundary}}(\mathbf{v}-\bar{\mathbf{v}})$, to the left-hand side of the momentum equation, where χ_{boundary} is the characteristic function of the boundary, and $\eta \ll 1$:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\,\mathbf{v} + \nabla p + \frac{1}{\eta}\chi_{\text{boundary}}(\mathbf{v}-\bar{\mathbf{v}}) = \mathbf{f} \tag{II.11}$$

In practice we use $\eta = 10^{-20}$, consult [4] for a study of the effect of η. As discretization is not addressed in the original references [4, 61], we derive here how to discretize the penalization term to allow for arbitrarily large time steps. In particular an implicit discretization is required, as an explicit discretization restricts the time step to be in the order of $\Delta t \leq 2\eta$, which is highly unpractical. This can be seen by performing a Von Neumann analysis of the simplified equation $\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\eta}\mathbf{v} = 0$. We describe first how to discretize the penalized momentum equation and how it affects the Poisson equation. Finally we consider how to discretize the penalized guiding equations.

Outside boundaries the discretization is equal to the non-penalized discretization since the characteristic function $\chi_{\text{boundary}} = 0$. Inside the boundary we discretize the velocity $\mathbf{v}$ of the penalization term at time $n+1$, thus obtaining the discretization of the penalized momentum equation Eq. (II.11):

$$\begin{aligned} \frac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} - \mathbf{f}^n &= -(\mathbf{v}^n\cdot\nabla)\mathbf{v}^n - \nabla p^n - \frac{1}{\eta}(\mathbf{v}^{n+1}-\bar{\mathbf{v}}^{n+1}) \\ &\Downarrow \\ \mathbf{v}^{n+1} &= \left(\tilde{\mathbf{v}}^n + \Delta t\left(-\nabla p^n + \frac{1}{\eta}\bar{\mathbf{v}}^{n+1}\right)\right)\frac{1}{1+\frac{\Delta t}{\eta}} \end{aligned} \tag{II.12}$$

where $\tilde{\mathbf{v}}^n = \mathbf{v}^n + (\mathbf{f}^n - (\mathbf{v}^n\cdot\nabla)\mathbf{v}^n)\,\Delta t$. The usual Poisson equation arises by taking the divergence of the momentum equation subject to the constraint $\nabla\cdot\mathbf{v}^{n+1} = 0$. However, taking the divergence of Eq. (II.12), subject to the same constraint, leads to a slightly different system of equations. Combining with the normal Poisson equation outside boundaries we obtain the following system of equations that should be used whenever boundaries are present:

$$\nabla\cdot\hat{\mathbf{v}}^n = \nabla\cdot\left(\chi_{\text{boundary}}\,\psi\nabla p + (1-\chi_{\text{boundary}})\nabla p\right) \tag{II.13}$$

and the velocity update

$$\mathbf{v}^{n+1} = \hat{\mathbf{v}}^n - \chi_{\text{boundary}}\,\psi\nabla p - (1-\chi_{\text{boundary}})\nabla p \tag{II.14}$$

respectively, where $\psi = \frac{1}{1+\frac{\Delta t}{\eta}}$ and $\hat{\mathbf{v}}^n = (1-\chi_{\text{boundary}})\tilde{\mathbf{v}}^n + \chi_{\text{boundary}}\left(\tilde{\mathbf{v}}^n + \frac{\Delta t}{\eta}\bar{\mathbf{v}}^{n+1}\right)\psi$. Note that we have left out the factor of Δt in front of p in both equations. This is possible since Δt is a constant and we are only interested in p up to a scale.

With these derivations in mind it is easy to devise a penalization strategy for the guiding equations. By setting $\mathscr{F} \equiv 0$ when its center point, $\mathbf{r}$, is inside a boundary, it is straightforward to verify that the following equations are equivalent to the guiding equations Eq. (II.7) and Eq. (II.8) outside the boundary, and equal to Eq. (II.13) and Eq. (II.14) inside the boundary. Due to the latter equality, the velocity is driven towards the actual boundary velocity inside the boundary. Consider G_1 to be the discretization of $\mathscr{G}_1$ and G_2 to be the discretization of $\mathscr{G}_2$, where $\mathscr{G}_1 = \frac{1}{\alpha(\mathbf{r})}\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\left[\mathscr{F} * \mathbf{v}\right](\mathbf{q})d\mathbf{q}$ and $\mathscr{G}_2 = \frac{1}{\alpha(\mathbf{r})}\int_\Omega (1-\alpha(\mathbf{q}))\mathscr{F}(\mathbf{q}-\mathbf{r})\mathbf{v}_{\text{low}}(\mathbf{q})$ are the guiding terms on the left- and right-hand side of Eq. (II.7). The penalized guiding equations become:

$$\nabla \cdot \hat{\mathbf{v}}^n = \nabla \cdot \left(\chi_{\text{boundary}} \psi \nabla p + (1-\chi_{\text{boundary}}) \frac{1}{\alpha(\mathrm{r})} \nabla p + G_1 - G_2 \right)$$

and

$$\mathbf{v}^{n+1} = \hat{\mathbf{v}}^n - \chi_{\text{boundary}} \psi \nabla p - (1-\chi_{\text{boundary}}) \frac{1}{\alpha(\mathrm{r})} \nabla p - G_1 + G_2$$

These equations replace Eq. (II.7) and Eq. (II.8) whenever boundaries are present.

6 Filter Estimation, Upsampling and Downsampling

For some types of simulations we vary the filter, $\mathscr{F}$, throughout the domain to achieve a certain artistic goal (see section 8). For physically based guiding velocity fields, it is in our experience very hard to apply a common lowpass filter, such as a Gaussian, by heuristically determining a suitable standard deviation. In particular it often results in guided simulations that are too smooth. Instead we construct the lowpass filter based on the following intuition: To use a high resolution velocity field as input to our guiding algorithm, the velocity field would have to be smoothed and downsampled. The guiding algorithm would then subsequently reconstruct the velocity field in high resolution and use this as $\mathbf{v}_{\text{low}}$ in the minimization. Assuming that the combination of these operations can be expressed as a convolution, the estimation of the lowpass filter, $\mathscr{F}$, encompasses the effects of smoothing (with a Gaussian kernel), G, downsampling, $\mathscr{D}$, and upsampling, $\mathscr{U}$. Essentially the desired filter is just $\mathscr{U} \circ \mathscr{D} \circ G$. However simply constructing the discrete filter in this way is very sensitive to how G is centered relative to $\mathscr{D}$. Another issue is what exactly the standard deviation of G should be set to. For this reason we estimate an approximation to $\mathscr{U} \circ \mathscr{D} \circ G$ offline via an optimization process. The estimated lowpass filter is not simulation dependent but depends only on the upsampling factor, the width, w, of the filter as well as the downsampling and upsampling methods.

The input to the filter estimation consists of a filter width, w, a 2D image, A, of N uniformly distributed random numbers (to avoid bias), and an upsampling factor. An optimization is then performed to find the standard deviation, σ, of G and the lowpass filter, $\hat{F}$, that minimizes the difference between $\mathscr{U}(\mathscr{D}(G(\sigma) * A))$ and $\hat{F} * A$, where $*$ denotes a convolution. In each iteration of the optimization, we construct a $N \times (w^2)$ system of linear equations, $M\mathbf{x} = \mathbf{b}$, that is solved in a least squares sense. The vector, $\mathbf{x}$, represents the filter, $\hat{F}$, we are estimating, while the right-hand side, $\mathbf{b}$, contains $\mathscr{U}(\mathscr{D}(G(\sigma) * A))$. The matrix, M, contains a row for each grid point in A,

and each row contains the values of A that fall in the $w \times w$ support of the filter we are estimating. To facilitate speedups during the multigrid solve of the guiding equations, we make the final guiding filter, F, separable. We have observed that $\hat{F}$ is usually very close to separable, so we construct the w diagonal values of F by setting them equal to the diagonal values of $\hat{F}$. The diagonal values completely define a separable F, as the values of F are given by a tensor product of the square root of the diagonal values with themselves. This tensor product definition directly extends to three dimensions, and we construct our filters for 3D guided simulations by a triple tensor product.

We both up- and down-sample by means of a cubic convolution [56]. Practical experience has shown that using linear interpolation is not sufficient. Boundaries are handled by explicitly setting the velocity of the low resolution boundaries in the low resolution velocity field before upsampling. In this way the upsampling process itself is unaware of boundaries.

7 Multigrid Solver

We adopt the multigrid method (see [14] for an introduction) to solve the linear system in Figure 3. Some of the advantages of the multigrid method are that for a grid with N grid points, it requires linear $O(N)$ storage, handles both symmetric and asymmetric linear systems and the low frequencies of the solution itself act as a preconditioner. Briefly explained, the multigrid method solves a linear system $A\mathbf{x} = \mathbf{b}$ on progressively coarser grids. It transfers solutions from coarser to finer grids and residuals from finer to coarser grids by means of interpolation and restriction operators, respectively. On each grid, or level, of the multigrid solve, a matrix operator is constructed and a relaxation method is employed for a small number of iterations. The strength of the multigrid method comes from the fact that the low frequencies of the solution converge much faster on coarser than finer grids.

For our linear system we found that extending the solver with restriction and interpolation operators that operate on cell faces as well as cell centers (in order to maintain a staggered grid on each level of the multigrid solve) improved the ratio of errors in each iteration roughly by a factor of two and required only a few multigrid cycles to converge.

The matrix operators on each multigrid level are stored in a compact storage format motivated by the fact that the stencils of the discretized equations (see Figure 3) are identical in many grid points. For example, in the case where $\mathscr{F}$ and α do not vary spatially, all grid points, that contain fluid and for which the support of the stencil is completely inside the spatial domain, will have identical stencils at the top multigrid level. By means of interpolation and restriction operators this property transfers to lower levels as well. Our matrix storage format is comprised of an indexed representation in which each grid point stores an index into an array of unique stencils, such that no duplicate stencils are stored. When the indexed representation is constructed, a stencil in a given grid point is only computed whenever it is detected that the stencil in the current grid point may not be identical to any of the stencils previously computed, thus speeding up the computation. Whenever $\mathscr{F}$ and α do not vary spatially it

is possible to further reduce storage requirements, because $\mathscr{F}$ is separable, meaning that $\int_\Omega \mathscr{F}(\mathbf{q}-\mathbf{r})\,[\mathscr{F} * \mathbf{v}]\,(\mathbf{q})d\mathbf{q}$ is also separable. Hence, only a $1D$ representation of the stencil needs to be stored. Currently we only exploit this on the top level. We do not even store the matrix operator at the topmost multigrid level when separability can be exploited because we found that both matrix multiply and relaxation can be computed efficiently, in parallel, on the fly. For the 256^3 guided smoke spray example (see section 8 and Figure 1) using a lowpass filter of width 9, the combined effect of our indexed representation and separability reduced total storage requirements of the matrix operators to 208MB from the 4.04TB required if storing all non-zero entries. In the case of moving boundaries, the stencils at the grid points that change status from boundary to fluid or vice versa are modified by the penalization method. Although in the worst case the entire matrix has to be re-computed when this happens, in practice only a few grid points, compared to the total number of grid points in the domain, change status. Hence the matrix can be re-computed locally by only updating the stencil values that are actually affected by the change. The locality of the change is maintained throughout lower multigrid levels, hence facilitating a fast update to matrix operators at all levels.

We have taken the approach of parallelizing the individual components of our multigrid solver such as the relaxation and matrix-vector multiplication using Intel Thread Building Blocks [104]. This gives a speedup of between four and five on an eight-core machine. Whenever possible, the relaxation and matrix-vector multiplication operate in parallel using separable filters only. We use Successive Over-Relaxation (SOR) with an over-relaxation parameter of 1.35 (found experimentally). Direct parallelization of SOR cannot guarantee a data flow consistent with a serial SOR algorithm, and hence the convergence analysis of the serial algorithm does not apply. However, we found that parallelizing SOR gives convergence rates equivalent to those obtained in serial. Particularly these convergence rates are significantly better than those obtained using weighted Jacobi relaxation.

8 Results and Discussion

We illustrate our variational guiding approach with the following examples. All simulations were run on a MAC Pro with 4GB of memory and two Intel Xeon quad-core 2.80GHz processors. We employ BFECC [57] and monotonic cubic interpolation [26] for advection. Identical time steps were used in low and high resolution.

Rising Smoke Column

A spherical density- and hot temperature-source is placed at the bottom of an elongated domain in which the smoke velocities are subjected to buoyancy. Both the density and temperature fields are advected in the flow. The low-resolution domain is $16 \times 64 \times 16$, and the high-resolution domain is upsampled by a factor of four to $64 \times 256 \times 64$. Figure 4 illustrates that the low-resolution simulation rises slower and with less turbulence than the high-resolution simulation. Due to diffusion of densities the smoke column in low resolution appears to be thicker than in high resolution. The rightmost image shows that just using the simple approach of upsampling the low-

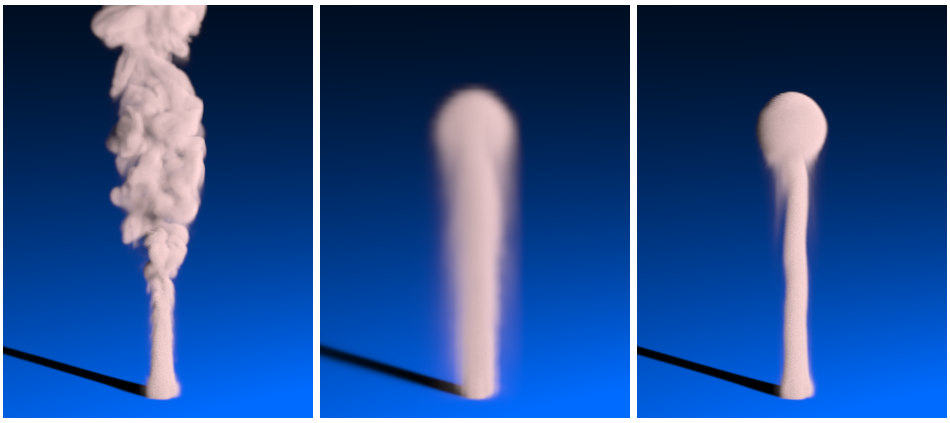


Figure 4 Left: High resolution simulation. **Middle:** Low resolution simulation. **Right:** Low resolution velocity field upsampled to high resolution followed by pressure projection and density advection (note that this is not a guided simulation). High frequencies cannot develop during the standard pressure projection [12].

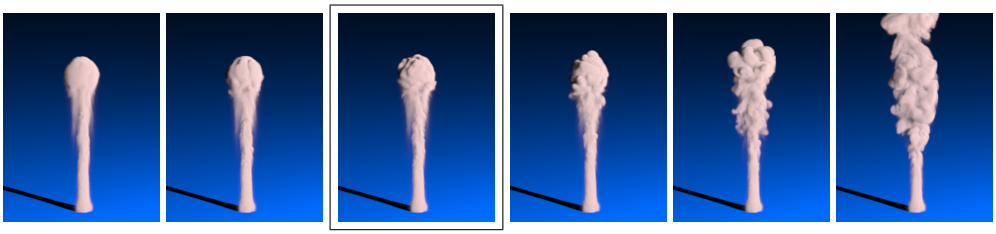


Figure 5 This figure illustrates a smooth interpolation from a strictly guided to an unguided simulation by means of varying the α parameter. Left to right: $\alpha = 0.15, 0.45, \mathbf{0.65}, 0.85, 0.97, 1.0$. We use $\alpha = 0.65$ for guided simulations.

resolution velocity field does not result in high frequency details. Guided simulations with varying α are shown in Figure 5 and enable a smooth interpolation between a strictly guided and an unguided simulation. In practice we run guided simulations with $\alpha = 0.65$ which appears to obtain a plausible mix of high frequency detail and guided low frequency flow. Increasing the width of the guiding filter allows for higher frequency detail to develop and we use a 9^3 filter stencil. The guiding matrix operators required 72MB and took 119 seconds to compute. 4 multigrid cycles were required to converge to a divergence of roughly 10^{-5} for the guided (31 seconds per frame) and unguided (16 seconds per frame) simulations at resolution $64 \times 256 \times 64$. Figure 6 illustrates that for Eulerian smoke simulations, undesirable features may develop over time when explicitly blending the low frequencies of the current velocity field with the upsampled guiding velocity field as proposed in [123]. Granted, their method is based on guiding particles whereas we set the low frequencies everywhere in the domain. Additionally, the choice of blend-factor used in [123] may, in our experience, give plausible results for one simulation, but undesired features for another. This implies that manual experimentation in high resolution may be required. However, if a successful blend-factor can be found, this simple method provides a faster alternative to our method.

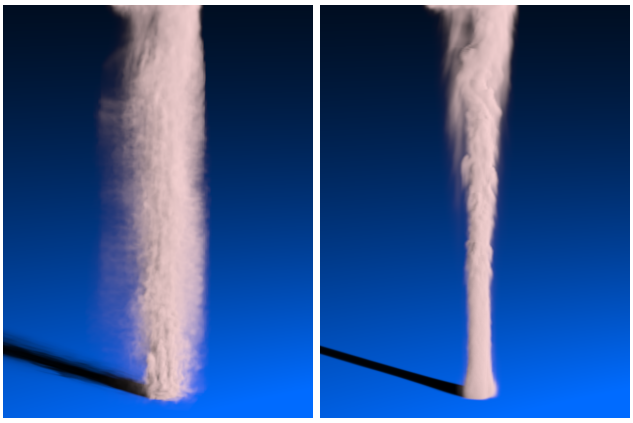


Figure 6 Frame 399 from the smoke column simulation. **Left:** Explicitly blending with a low frequency guiding velocity field before pressure projection may result in undesirable features developing over time. **Right:** Guided simulation with identical blending parameter, α, for comparison.

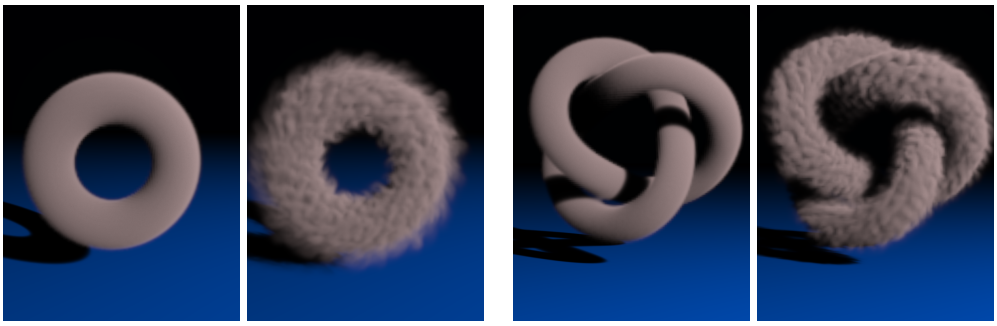


Figure 7 **Left:** A low-resolution, non-physically based circular velocity field is used to evolve a smoke torus. Both $\mathscr{F}$ and α vary spatially to facilitate the desired motion while preserving the shape of the torus. **Right:** A similar setup with a trefoil knot.

Rising Smoke Column with Moving Boundary

This setup is similar to the Rising Smoke Column simulation except that a moving boundary is timed to touch the top of the smoke column in low resolution as shown in Figure 8. In high resolution the boundary passes through the faster moving smoke. The guided high-resolution simulation retains the qualitative features of the low-resolution simulation and adds higher frequency detail. The initial guiding matrix operators required 280MB and took 254 seconds to compute. The guided and unguided high resolution simulations took respectively 84 and 45 seconds per frame.

Smoke Spray

A velocity-, density- and temperature-source injects hot high-velocity smoke subject to a buoyancy force as shown in Figure 1. Again, the guided simulation retains the low frequency qualitative features of its guiding simulation. The total storage required for the matrix operators of the guided simulation is 208MB and the computation took 696 seconds. The guided simulation took 580 seconds per frame and the unguided simulation 315 seconds at resolution 256^3.

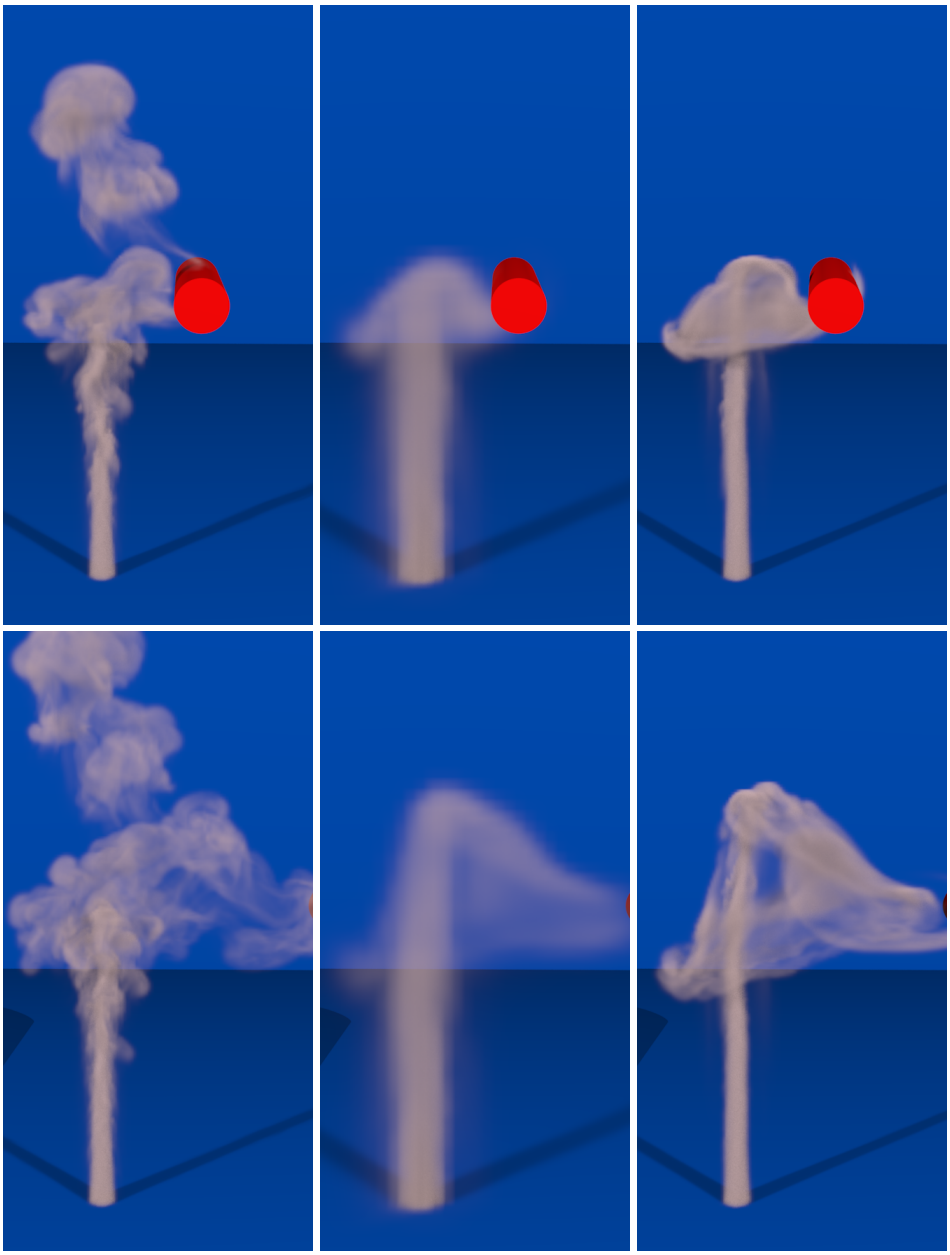


Figure 8 Upper triple: High resolution unguided, low resolution unguided and high resolution guided simulations at frame 210 of a smoke column interacting with a moving boundary. In low resolution and guided high resolution the boundary touches the top of the smoke column whereas in high resolution the boundary passes through the smoke. Due to diffusion the column appears thicker in low resolution. **Lower triple:** Frame 256 from same simulation.

Guiding Curves

A low-resolution, non-physically based velocity field in the shape of a torus is used to guide a smoke torus (see Figure 7). Both the lowpass filter $\mathscr{F}$ and the guiding parameter α vary spatially; the identity filter is used in a narrow band inside the initial smoke band to prevent drift of the low-frequency velocities, and guiding is gradually

diminished outside the initial smoke band by increasing α. Vorticity confinement[26] is used to induce high-frequency instabilities. The rightmost images of Figure 7 show the same setup with a trefoil knot curve. For both simulations it was necessary to uniformly quantize the values of α to limit the number of unique stencils in the matrix operators and thereby the storage requirements. We have verified experimentally that the quantization did not cause visual artifacts. For the torus simulation, the matrix operators required 1496 MB and took 1061 seconds to compute. The simulation ran for 150 seconds per frame at resolution 128^3. The trefoil knot simulation needed 2064 MB, and the computation of the matrix operators took 2549 seconds. The simulation took 155 seconds per frame at resolution 128^3.

The framework proposed in this paper has several limitations:

- It is an assumption of our method that an artist can obtain a satisfactory bulk movement in low resolution. If this is not the case, our method does not apply.
- Provided that an appropriate low-resolution simulation can be found, our framework allows animators to prototype in low resolution and then add details with a guided high-resolution simulation. However, for the examples presented in this paper we found the guided simulations to incur an overhead of $84-94\%$ compared to the high-resolution unguided simulation. For this reason we wish to pursue further strategies for optimization in the future.
- The amount of density-diffusion and -dissipation is significantly higher in low resolution. For this reason, features in the high-resolution guided simulation may deviate in intensity from the low-resolution simulation. Specifically, this is the case when sheets of diffused densities develop and when thin structures dissipate away in low resolution.
- Our approach fails if boundaries have very thin or fine features that cannot be represented simultaneously on the high- and low-resolution grids. At least this will be the case close to the boundary where boundary conditions must be fulfilled. We plan to address this in future work, possibly by adapting the methods for handling thin shells by Guendelman *et al.* [40] and the accurate, variational boundary coupling of Batty *et al.* [6].

9 Conclusion

Smoke simulations are notoriously hard to art-direct due in part to numerical viscosity and the non-linearity of the governing equations. In this paper we proposed a self-consistent variational method for guiding a high resolution Eulerian smoke simulation by a lower resolution flow. The low frequencies of the guided simulation agree as much as possible with the low resolution flow, and higher frequencies can develop. It is a fundamental assumption of our method that a satisfactory low-resolution simulation exists, otherwise our method does not apply. The low-resolution flow can originate from a physically based simulation, be purely art directed or arise from a combination of the two. Our method allows for a smooth interpolation between a strictly guided

and an unguided physical simulation, and appears to be free of the visual artifacts that may result from using previous methods. An interesting alternative to the multigrid approach taken in this paper might be the multilevel bases described by Oswald [100]. Finally, we believe that the variational approach taken in this paper is applicable to a wide range of fluid control paradigms in computer graphics.

ACKNOWLEDGEMENTS

This work was partially funded by the Danish Agency for Science, Technology and Innovation. We thank Ole Østerby for advice on numerics, Michael Clive and Ryo Sakaguchi for helping define the problem, our colleagues at Digital Domain for their support, and the reviewers, committee and program chairs for constructive feedback.

PAPER III

IMPROVED VARIATIONAL GUIDING OF SMOKE ANIMATIONS

Michael Bang Nielsen Brian Bunch Christensen

Abstract

Smoke animations are hard to art-direct because simple changes in parameters such as simulation resolution often lead to unpredictable changes in the final result. Previous work has addressed this problem with a guiding approach which couples low-resolution simulations – that exhibit the desired flow and behaviour – to the final, high-resolution simulation. This is done in such a way that the desired low frequency features are to some extent preserved in the high-resolution simulation. However, the steady (*i.e.* constant) guiding used often leads to a lack of sufficiently high detail, and employing time-dependent guiding is expensive because the matrix of the resulting set of equations needs to be recomputed at every iteration. We propose an improved mathematical model for Eulerian-based simulations which is better suited for dynamic, time-dependent guiding of smoke animations through a novel variational coupling of the low- and high-resolution simulations. Our model results in a matrix that does not require re-computation when the guiding changes over time, and hence we can employ time-dependent guiding more efficiently both in terms of storage and computational requirements. We demonstrate that time-dependent guiding allows for more high frequency detail to develop without losing correspondence to the low resolution simulation. Furthermore, we explore various artistic effects made possible by time-dependent guiding.

Published as: Michael B. Nielsen and Brian B. Christensen. Improved Variational Guiding of Smoke Animations. In *Computer Graphics Forum, Volume 29, Number 2, 2010 / Proceedings of the Eurographics Conference 2010*, (Norrköping, Sweden, May 3–7, 2010). Eurographics 2010. Eurographics, Goslar, Germany. *To appear.*

1 Introduction

Creating believable fluid effects for computer graphics typically requires the use of advanced and time-consuming computational techniques. For many years the exploration and development of such techniques have been the subject of the computational fluid dynamics (CFD) community. However, whereas numerical and physical accuracy are paramount in CFD, visual plausibility [26], low simulation cost [114] and artistic control [32] are important for computer graphics.

Today, high-resolution fluid simulations that contain sufficient detail for visual effects have turnaround times of several hours or even days. Such simulation times are infeasible for effects-artists during the design stage where near-interactive feedback times are required. For this reason artists typically prototype simulations at low resolution. Once a certain design has been settled for, they increase the resolution to obtain the level of detail required for the shot. However, this approach is not always successful as a change in resolution may not only incur an addition of high frequency features but may completely change the low frequency properties of the flow that the artist settled for in low resolution. Numerical viscosity originating from the discretization is one of the primary reasons for this. This has led several authors to propose algorithms for obtaining better correspondence between low- and high-resolution simulations. This has been done by allowing low-resolution simulations to *guide* higher resolution simulations [7, 92, 123] and by ensuring that properties such as kinetic energy are preserved in the discretization [79].

In this paper we explore time-dependent – as opposed to steady – guiding of smoke animations. Specifically we show that time-dependent guiding can be used for creating dynamic artistic effects and allows for more high frequency detail to develop in high resolution compared to previous work [92]. We take an approach similar to that of Nielsen *et al.* [92], where guiding is formulated as a constrained minimization problem. However, Nielsen *et al.* studied only steady guiding effects and it turns out that their mathematical formulation requires the matrix resulting from their Euler-Lagrange equations to be re-computed, whenever the guiding strength varies in time. In particular, the spatially varying scalar field used to control the guiding strength is by construction included in their matrix. Since this scalar field may change in large fractions of the domain in the case of time-dependent guiding, their matrix re-computation becomes costly when combined with the large stencils required by guiding. Typically their matrix computation time in the presence of spatially varying guiding weights is about an order of magnitude longer than the simulation time per frame. In addition their matrix storage requirements depend on the variation in the scalar field of guiding weights.

We propose a new mathematical model for guiding smoke animations at high resolution by means of animations at lower resolution. Our mathematical model separates low frequencies from high frequencies in the flow which leads to a set of Euler-Lagrange equations in which the resulting matrix is independent of the scalar field used to control the guiding strength. This means that a costly matrix re-computation does not have to be performed whenever the guiding changes in time. In addition our matrix often takes about an order of magnitude less time to compute and requires an

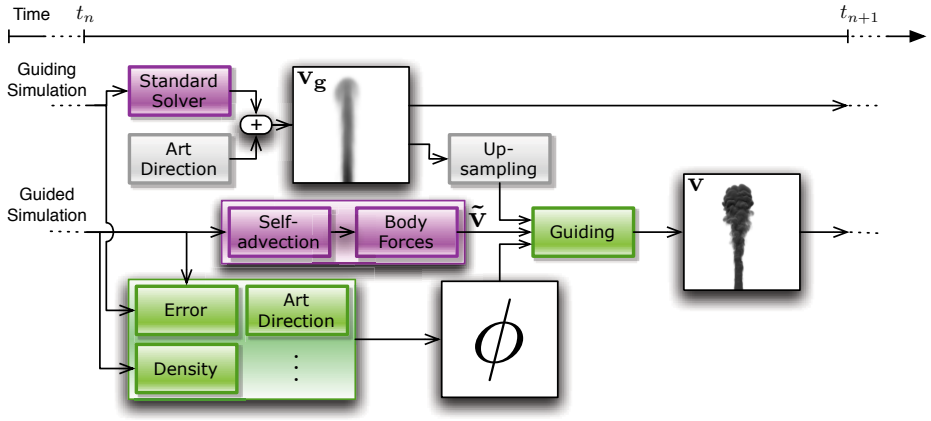


Figure 1 One simulation iteration using our method. A traditional fluid simulation pipeline (in purple) is equipped with a modified pressure projection step . Components which constitute our contributions are highlighted in green.

order of magnitude less storage than [92].

Our framework is based on the inviscid Euler equations $\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\,\mathbf{v} = -\nabla p + \mathbf{f}$ and $\nabla \cdot \mathbf{v} = 0$, where $\mathbf{v}$ is the velocity of the fluid, p denotes pressure, and $\mathbf{f}$ represents external forces. We solve these equations using the operator splitting approximation first introduced to computer graphics by Stam [114]. This method solves for self-advection, body forces, and incompressibility separately. Our model only involves a replacement of the usual Poisson equation and velocity update in the pressure projection step which solves for incompressibility. This is illustrated in figure 1 which shows one iteration of our guiding framework: The low-resolution guiding velocity field $\mathbf{v_g}$ is obtained using a traditional simulation pipeline (in purple) or through art direction. The guiding weights ϕ are then computed as explained in section 5. Subsequently, the guided high-resolution velocity field is advected and body forces are added to obtain an intermediate velocity field $\tilde{\mathbf{v}}$. Finally, our modified pressure projection step uses $\tilde{\mathbf{v}}$, ϕ, and an upsampled version of $\mathbf{v_g}$ to obtain the new high-resolution guided velocity field $\mathbf{v}$. To sum up, the contributions of this paper are:

- A novel mathematical model that leads to more efficient time-dependent guiding of smoke animations.

- Exploration of time-dependent guiding for artistic effects and for increasing the amount of high frequency features in the high-resolution guided flows.

An inherent assumption in our work is that the bulk movement of the flow can be satisfactorily represented at low resolution. Furthermore, our method cannot be used to specifically design the high frequency features, but on the other hand provides a starting point for iterating on higher frequency features.

2 Related Work

Controlling Navier-Stokes based simulations at a higher level than parameter tweaking is an idea that was first introduced by Foster and Metaxas [32]. More recently Treuille *et al.* [126] proposed a key-frame control framework for smoke simulations which uses a gradient descent based optimizer. In particular they optimize for forces which make the smoke assume key-framed poses. Later McNamara *et al.* [75] made this approach faster by employing the adjoint method.

Shi *et al.* [110] also consider key-frame control but focus on liquid simulations. They suggest a force-based solution which allows the liquid to follow rapidly changing target shapes. Other approaches similar to Shi *et al.* have considered the problem of making smoke follow target shapes [25, 44]. Finally, Kim *et al.* [59] propose a method for constructing path-guided smoke trails.

Guiding of liquid simulations based on the Lattice Boltzmann Method and Smoothed Particle Hydrodynamics has been demonstrated by Thürey *et al.* [123]. Their method is based on controlling the low frequencies of simulations through forces which are applied near Lagrangian control particles. Nielsen *et al.* [92] show that applying the idea of blending low frequency components of a simulation with a guiding velocity field before the pressure projection step may lead to noise developing over time for Eulerian smoke simulations. To avoid such artifacts they take the approach of formulating guiding as a constrained minimization problem. Concurrently, Mullen *et al.* [79] developed energy preserving integrators, which can also be employed to achieve higher correspondence between simulations at low and high resolution.

Contrary to the above guiding approaches where the high frequency detail arises from a physically based simulation, a lot of work has recently focused on procedural methods that add high frequency detail to a low-resolution input simulation [58, 86, 102, 107]. Common to these methods is that they improve both the run times and storage requirements over a full fluid simulation. The goal of Narain *et al.* [86] and Pfaff *et al.* [102] is essentially different from ours in that the synthesized flow should resemble the high-resolution flow, as opposed to the low-resolution flow, as well as possible. Narain *et al.* do note that the coupling from high frequencies to low frequencies can be disabled in order to retain the frequencies of the low resolution flow, but do not focus on this scenario. On the other hand, Schechter and Bridson [107] and Kim *et al.* [58] retain the low frequency flow and procedurally add high frequency detail as a post-process. As noted by [107] stable, laminar structures can be forced turbulent, a property shared by [58], whereas Narain *et al.* appear to have solved this problem. Of the above methods, only Pfaff *et al.* consider turbulent structures emanating from interaction with boundaries at high resolution. However, their method does not guarantee correspondence between the low- and high-resolution flows. Our proposed guiding method inherits the ability from the underlying physical simulation to automatically develop instabilities, including those arising in the vicinity of boundaries.

3 GUIDING

In this paper we define *guiding* as follows: An input velocity field, $\mathbf{v_g}$, is used to enforce a correspondence between the low frequencies of a higher-resolution, simulated velocity field, $\mathbf{v}$, and $\mathbf{v_g}$ itself. The input velocity field, $\mathbf{v_g}$, can originate from a physically based simulation or be constructed by an artist by any other means such as illustrated in figure 5. The goal of this section is to formulate guiding as a constrained minimization problem. The *minimization* will enforce the guiding, whereas the *constraint* will enforce that the resulting velocity field is divergence free. By employing calculus of variations and the method of Lagrange multipliers, the minimization problem is transformed into a set of partial differential equations, the Euler-Lagrange equations, that solve for the corresponding stationary point. In this paper we will denote this set of partial differential equations the *guiding equations*. Note that the Poisson equation solving for incompressibility in a standard fluid solver can in fact also be derived from a constrained minimization problem [29, pp. 202-204].

Ideally, the guiding should only affect the low frequencies of $\mathbf{v}$. Therefore we separate the low and high frequencies in our mathematical model. In addition, this separation leads to guiding equations where the left-hand-side, *i.e.* the matrix, is independent of the spatially and temporally varying guiding weights which we denote ϕ. $\phi(\mathbf{x},t) \in [0;1]$ is a time-dependent scalar field that enforces the strength of the guiding. If $\phi = 1$ in a given point, guiding is enforced with maximal strength (see second image from the left in figure 3). If on the other hand $\phi = 0$, the guiding equations essentially reduce to the normal Poisson equation and velocity update, hence no guiding is enforced. The scalar field ϕ is computed as described in section 5 prior to solving the guiding equations for pressure and velocity.

We first state our mathematical model and then elaborate on an intuitive interpretation below. In particular we wish to minimize

$$\int_\Omega (1-\phi(\mathbf{x}))\,|[\mathscr{F} * \mathbf{v}](\mathbf{x}) - [\mathscr{F} * \tilde{\mathbf{v}}](\mathbf{x})|^2\,d\mathbf{x} + \tag{III.1}$$

$$\int_\Omega \phi(\mathbf{x})\,|[\mathscr{F} * \mathbf{v}](\mathbf{x}) - \mathbf{v_g}(\mathbf{x})|^2\,d\mathbf{x} + \tag{III.2}$$

$$\int_\Omega |[\mathbf{v} - [\mathscr{F} * \mathbf{v}]](\mathbf{x}) - [\tilde{\mathbf{v}} - [\mathscr{F} * \tilde{\mathbf{v}}]](\mathbf{x})|^2\,d\mathbf{x} \tag{III.3}$$

subject to the constraint

$$\nabla \cdot \mathbf{v}(\mathbf{x}) = 0 \tag{III.4}$$

where Ω is the fluid simulation domain, $\tilde{\mathbf{v}}$ is the current high resolution velocity field in the operator splitting sequence (*i.e.* the velocity field from the previous frame updated by self-advection and forces), $\mathscr{F}$ is a lowpass filter, and $*$ is the convolution operator. Contrary to the approach taken in [92], we separate the low and high frequencies in the minimization problem. Specifically, expressions (III.1) and (III.2) of the minimization problem involve the low frequencies of $\mathbf{v}$, expressed as the convolution $[\mathscr{F} * \mathbf{v}]$, whereas (III.3) involves the remaining high frequencies, $\mathbf{v} - [\mathscr{F} * \mathbf{v}]$, of $\mathbf{v}$. Expression (III.1) (expression (III.2)) contains the low frequencies of the difference between the current high resolution velocity field (the guiding velocity field) and the velocity field we are solving for. Expression (III.3) on the other hand contains the high frequencies

of the difference between the current high resolution velocity field and the velocity field we are solving for. By keeping (III.3) independent of the guiding weights it turns out that the resulting matrix also becomes independent of the guiding weights. Finally, Eq. (III.4) is the constraint that forces the resulting velocity field to be divergence free.

If $\phi = 0$, (III.2) vanishes, and the minimization problem reduces to a minimization of the difference between $\mathbf{v}$ and $\tilde{\mathbf{v}}$. Furthermore, if $\phi = 0$ and $\mathscr{F}$ is a perfect low-pass filter, (III.1) and (III.3) can be combined into $\int_\Omega |\mathbf{v}(\mathbf{x}) - \tilde{\mathbf{v}}(\mathbf{x})|^2 d\mathbf{x}$ which can be seen by transforming to Fourier space and applying Parseval's theorem. Minimizing $\int_\Omega |\mathbf{v}(\mathbf{x}) - \tilde{\mathbf{v}}(\mathbf{x})|^2 d\mathbf{x}$ subject to the constraint Eq. (III.4) is in fact equivalent to solving the usual Poisson equation and subsequent velocity update [29]. Hence, the guiding equations will essentially result in an unguided simulation for $\phi = 0$.

If $\phi = 1$, the low frequencies of the resulting velocity field $\mathbf{v}$ will minimize the difference to $\mathbf{v_g}$ alone and hence only the guiding will be in effect, see (III.2). On the other hand $\phi \in (0;1)$ will minimize the difference to both $\mathbf{v_g}$ and the low frequencies of $\tilde{\mathbf{v}}$, and thus $\mathbf{v}$ will be a blend of the two low-frequency components. It is an important point of this paper that relaxing the strength of the guiding by lowering ϕ whenever possible is key to allowing more turbulence to develop naturally in guided simulations.

To turn our proposed minimization problem into the corresponding Euler-Lagrange equations (the guiding equations) we employ calculus of variations. The full derivation is included in appendix A. Here we merely present the result and explain its consequences. Specifically, the constrained minimization problem (III.1) – (III.4) leads to the following equations:

$$\begin{aligned} &\mathbf{v}(\mathbf{x}) + 2\int_\Omega \mathscr{F}(\mathbf{y}-\mathbf{x})\left[\left[\mathscr{F} * \mathbf{v}\right](\mathbf{y}) - \mathbf{v}(\mathbf{y})\right] d\mathbf{y} + \nabla p(\mathbf{x}) = \\ &\tilde{\mathbf{v}} + \int_\Omega \mathscr{F}(\mathbf{y}-\mathbf{x})\left[(2-\phi)\left[\mathscr{F} * \tilde{\mathbf{v}}\right](\mathbf{y}) - 2\tilde{\mathbf{v}}(\mathbf{y}) + \phi \mathbf{v_g}(\mathbf{y})\right] d\mathbf{y} \end{aligned} \tag{III.5}$$

$$\nabla \cdot \mathbf{v}(\mathbf{x}) = 0 \tag{III.6}$$

where p is the pressure of the fluid. There is one equation for each point $\mathbf{x}$ in the fluid simulation domain. All equations are solved simultaneously for both the guided velocity field, $\mathbf{v}$, as well as the pressure, p. On a staggered grid, Eq. (III.5) leads to one equation for each face, and Eq. (III.6) leads to one equation for each cell center. This amounts to $(D+1)N$ equations in $(D+1)N$ unknowns, where N is the number of grid points. Note that a standard fluid solver also has to solve for $(D+1)N$ unknowns, velocity and pressure. However, in this case there is only a one-way coupling between the unknown pressure and velocity and hence the Poisson equation involves only the pressure, N equations in N unknowns. The derived equations are amenable to solution in the form stated above. However, for solution using relaxation methods such as Jacobi or Gauss-Seidel (including the multigrid method which utilizes these relaxation methods) it is convenient to replace Eq. (III.6) by Eq. (III.5) with the divergence operator applied to both sides of the equation. By exploiting the constraint, Eq. (III.6), $\nabla \cdot \mathbf{v}(\mathbf{x})$ vanishes from the resulting equation and hence the constraint remains enforced. Furthermore, this will ensure non-zero diagonal entries as required by the relaxation methods. Eq. (III.5) and Eq. (III.6) share the properties of the model of [92] that if $\mathbf{v_g} = \mathscr{F} * \mathbf{v}$ and $\mathbf{v}$ is divergence free, the resulting velocity field will be

Figure 2 Close-ups of the smoke column simulation comparing from left to right the guiding method of Nielsen *et al.* [92], our method guiding with eroded densities and finally our method guiding with eroded densities combined with the error estimate.

$\mathbf{v}$ itself. Also, $\mathbf{v_g}$ does not have to be divergence free. In addition, our new model has the following properties not shared by [92]:

- The guiding weights, ϕ, appear only on the right-hand-side of the equations. This means that whenever ϕ depends on time, the matrix operators do not have to be re-computed.
- Spatially varying ϕ will not increase the storage requirements of the matrix operators since ϕ appears only on the right-hand-side, hereby also facilitating faster computation of the matrix operators.

4 Implementation

The guiding equations are discretized on a staggered grid with velocities sampled on cell faces and pressure sampled in cell centers [26]. We use BFECC [22] and monotonic, cubic interpolation [26] for advection. To solve the equations we employ a parallel multigrid implementation [14] combined with a simple compression scheme of the matrix operators similar to the one described in [92], except that we use a relaxation parameter of 0.9 in our SOR implementation. Boundaries are handled by applying the penalization method of Angot *et al.* [4, 61]. Finally, upsampling and lowpass filter estimation is handled as in [92].

5 Time-Dependent Guiding Effects and Results

Guiding is applied to enforce correspondence between the low- and high-resolution velocity fields. However, although our guiding model is separating the low and high frequencies, it is evident in practice that imposing too much guiding will constrain the high frequencies as well. On the other hand, relaxing the guiding weights uniformly

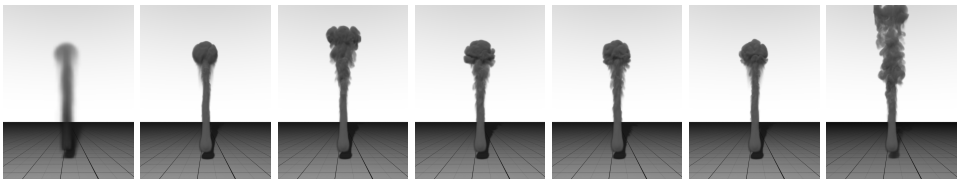


Figure 3 Identical frames from several simulations of a smoke column at resolution $64 \times 256 \times 64$ rising due to a buoyant force. The leftmost image depicts a low resolution simulation and the rightmost image depicts a high resolution simulation. The remaining images from left to right: (1) Our method with a uniform weight of $\phi = 1$. (2) Our method with a uniform weight of $\phi = 0.02$. The amount of turbulence along the column is similar to the non-uniform guiding in the image to the right. However, using uniform weights, the column rises faster than the low-resolution simulation. (3) Our method with eroded densities combined with the error estimate. $\phi_{low} = 0$ and $\phi_{high} = 0.35$. (4) Same approach with $\phi_{low} = 0$ and $\phi_{high} = 0.55$. (5) Same approach with $\phi_{low} = 0.15$ and $\phi_{high} = 0.35$. Notice that the relatively high ϕ_{low} results in a lack of high frequency detail.

will cause the guided simulation to behave increasingly as an unguided simulation, thereby developing more turbulent structures. However, this may come at the cost of losing correspondence to the low-resolution velocity field, see figure 3.

In the following we explore a number of different approaches to guiding the high-resolution simulation with both spatially and temporally dependent guiding weights, $\phi(\mathbf{x},t)$, *e.g.* to allow more turbulent structures to emerge. The guiding weights are represented as a time-dependent scalar field fed into the linear equation solver in each iteration. It is important to note that the high frequency features that develop in the guided simulations are instabilities arising in the underlying high-resolution, physically based simulation. Vorticity confinement or procedural turbulence can of course be added on top of the guided simulations to synthesize an even more turbulent behaviour.

5.1 Time-Dependent Guiding with Smoke Density

Using the smoke density as a guiding weight is based on the observation that in many cases we want thick volumes of smoke to adhere to the bulk motion prescribed by the low-resolution guiding velocity field. On the other hand we also want turbulence to develop in areas where the smoke is wispy. Let the smoke density be denoted by $d(\mathbf{x},t)$ and assume that $d(\mathbf{x},t) \in [0;1]$. We construct the guiding weights as $\phi(\mathbf{x},t+\Delta t) = \phi_{high} d(\mathbf{x},t) + \phi_{low}(1-d(\mathbf{x},t))$, where ϕ_{low} and ϕ_{high} are lower and upper bounds on the guiding weights, respectively. Unless stated otherwise we use $\phi_{low} = 0$ and $\phi_{high} = 0.35$ as found by experimentation, see figure 3. In our experience, guiding with density yields better results when combined with the extensions described below.

5.1.1 Combining with Erosion of Densities

We have explored performing an erosion of the smoke density used to construct the guiding weights. The idea behind this is that the guiding will be relaxed in a wider band

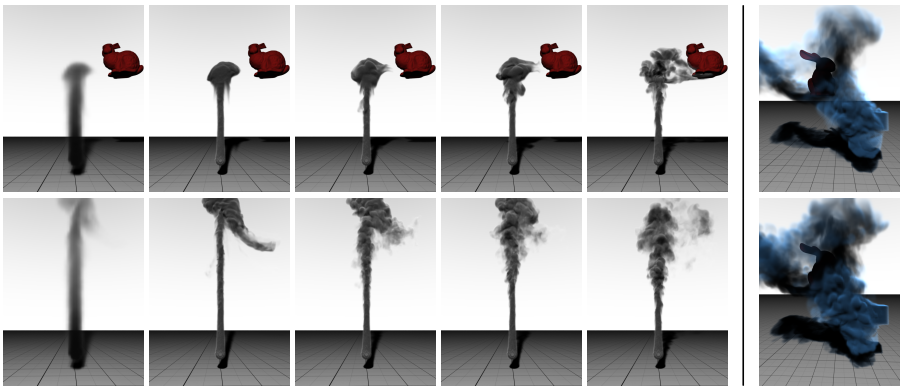


Figure 4 Left: A rising smoke column interacting with a translating Stanford bunny at resolution $128 \times 256 \times 64$. *Upper row:* Early frame. From left to right: The low resolution guiding simulation, the guiding approach of [92], our guiding with eroded densities, our guiding with eroded densities combined with the error estimate and finally the high resolution simulation. *Lower row:* Later frame. **Right:** Two guided simulations of a Stanford bunny rotating close to a smoke jet subject to a buoyant force at resolution 128^3. *Top:* The guiding method of [92]. *Bottom:* Our method guiding with eroded densities. The addition of turbulent features is more subtle and less noticeable in this case, but is visible for example in the upper left corner.

in regions between smoke and clear air, thereby allowing more turbulent structures to emanate in these areas. Erosion is characterized by a single parameter which is the width of a stencil centered at each grid point. The result of the erosion operation at a given grid point is the minimum density value found under the support of the stencil when centered at that particular grid point. We have found by experimentation that using a stencil width of three generates the best results. Using wider stencils tends to make the simulations behave more like unguided simulations, *e.g.* making smoke rise faster. Figure 2 shows an example of a simulation using the eroded density as guiding weight and compares it to uniform guiding weights throughout the domain using the model of Nielsen *et al.* [92]. In all comparisons to [92], we set their scaling parameter to $\alpha = 0.65$ as suggested in their paper. As is evident from the figure, more turbulence appears when the guiding weights are relaxed using our method. Additional examples are shown in figure 4.

Time-dependent guiding using [92] in the example of the rising smoke column in figure 2 requires roughly 1600 seconds per frame to compute the matrix which takes up roughly 1.6GB of storage. Using our guiding model the matrix computation takes roughly 120 seconds, requires 74MB, and only has to be performed once for an entire simulation. The guided smoke column simulation in figure 2 runs for 36 – 37 seconds per frame using our method, depending on which approach is utilized to compute the guiding weights. In contrast an unguided simulation at the same resolution takes roughly 17 seconds per frame. A similar result was reported in [92]. However, we emphasize that their results were obtained using steady guiding in which the guiding weights do not vary in time. If in fact the guiding weights vary in time, their method requires 1600 seconds per frame for the matrix computation plus the simulation time. In our fluid solver implementation we employ the relatively expensive combination of BFECC and cubic interpolation for high quality advection of both velocities, densities

and temperature. For the unguided simulation in figure 3, these advection steps take up 46% of the total time per frame, whereas the multigrid solver uses only 22% of the time per frame. The time required to solve the guiding equations in the same example is about 5.2 $\times$ the time required to solve the Poisson equation and perform the velocity update in a standard fluid solver at the same resolution. However, since the multigrid solver only takes up 22% of the total time per frame for an unguided simulation, this explains why the total time per frame is not more than doubled, despite the fact that more unknowns are involved in our linear system of equations.

5.1.2 Combining with an Error Estimate

It seems natural to consider relaxing the guiding whenever the error between the guiding velocity field and the low frequencies of the guided velocity field is low. We have explored this idea as follows: Let $e(\mathbf{x},t) = |[\mathscr{F} * \mathbf{v}](\mathbf{x},t) - \mathbf{v_g}(\mathbf{x},t)|^2$ be the error between the guiding velocity field and the low frequencies of the guided velocity field. Furthermore we denote by $\bar{e}(\mathbf{x},t)$ the error estimate normalized to a maximal value of one and construct the guiding weights as the function $\phi(\mathbf{x},t+\Delta t) = \phi_{high}\bar{e}(\mathbf{x},t)d(\mathbf{x},t) + \phi_{low}(1-\bar{e}(\mathbf{x},t)d(\mathbf{x},t))$. Incorporating the error estimate into the guiding weight works well for some types of simulations and typically generates more turbulent structures than guiding with smoke density alone (see figure 2). However, for other types of simulations, incorporating the error estimate causes the guiding to lose some correspondence to the low-resolution simulation as shown to the left in figure 4.

5.2 Time-Dependent Guiding with Curves

The previous examples have looked at guiding with a velocity field obtained from a low-resolution simulation. However, artistic effects can be achieved by constructing velocity fields through non-physically based methods such as modeling or painting. These can then be used *as is* or combined with an existing low-resolution simulation. They are applied using time-dependent guiding weights to enable artists to enhance, decrease or create new low frequency motion in an intuitive manner. We have examined this idea through the following approach: Let $\mathbf{v_{curve}}(\mathbf{x},t)$ be an artist-specified guiding velocity field and let $\phi_{curve}(\mathbf{x},t)$ be similarly specified guiding weights. Essentially these fields could describe any motion the artist desires, and in particular they do not have to ensure a divergence free guiding velocity field. We will specify ϕ such that guiding is only applied in a narrow tube around various curves $C_1,\ldots,C_n$. These same curves are used to create velocity fields $\mathbf{v_{curve}}_1,\ldots,\mathbf{v_{curve}}_n$ which flow along them. ϕ is animated over time by gently increasing and decreasing local guiding weights $\phi_{curve_1},\ldots,\phi_{curve_n}$ in order to gradually enable and disable the flows induced by the corresponding curves. These local guiding weights form tubes in which velocities along the curves are induced and they are illustrated in red in the leftmost pair of images of figure 5. In the animation we consider, the two guiding curves are enabled one after the other. As stated, these guiding effects can also be combined with error estimates and smoke densities to produce more high frequency detail. An example of this, where these additional effects are combined with the ϕ_{curve_i}, can be seen to the

right in figure 5. The matrix was computed in 181 seconds and required 82MB of storage at simulation resolution 128^3. Each frame required 65 seconds to simulate. Employing the guiding model of Nielsen *et al.* in this example required roughly 1800 seconds *per frame* to compute the matrix, and roughly 3GB to store it. Note again, that the matrix computation is only required once *per simulation* in our framework.

5.3 Discussion and Limitations

The computation time required for a guided high-resolution simulation is roughly doubled when compared to an unguided simulation at the same resolution. Procedural methods are faster and require less storage than a standard fluid solver. However, as opposed to our guiding approach, procedural methods that ensure a correspondence between low and high resolution can force otherwise stable laminar flow to become turbulent and do not capture the instabilities naturally occurring near boundaries. In the accompanying video we compare our guiding method to the wavelet turbulence method [58] to demonstrate these differences.

Guiding with the error estimate does not always result in significantly higher detail, and for some simulations the similarity between low and high resolution can become too weak. Another potential problem is that all grid points in the domain are coupled by the error estimate. This may be improved by using absolute errors combined with clamping and/or nonlinear interpolation. Furthermore, incorporation of the density field and error into the guiding weights does not currently ensure density matching between the low- and high-resolution simulations. In the future we wish to explore density matching which may be used to ensure similarity for guiding with the error estimate and ensure that no drift occurs over time. Note that [92] also does not prevent drift of densities, since their variational model considers only velocities. Additionally, exploring other guiding weights *e.g.* based on the gradient of density as opposed to density alone is an interesting direction for future work.

Finally, for some simulations a simpler method such as blending the low frequencies of the high resolution velocity field with the guiding velocity field might suffice [123], provided that artifacts do not appear within the time-frame required by the simulation.

6 Conclusion

Smoke animations are hard to art-direct because of numerical viscosity and the nonlinear nature of the governing inviscid Euler equations. In this paper we propose an improved variational method for guiding high-resolution smoke simulations using low-resolution velocity fields. The method is faster and requires less storage for time-dependent guiding than previous methods. We have demonstrated that time-dependent guiding can be used to achieve more high frequency detail and achieve artistic effects. In conclusion, we are convinced that the variational approach adopted in our method can be applied to a large number of fluid control paradigms. We also believe that further exploration into determining the guiding weights ϕ can yield new ways of

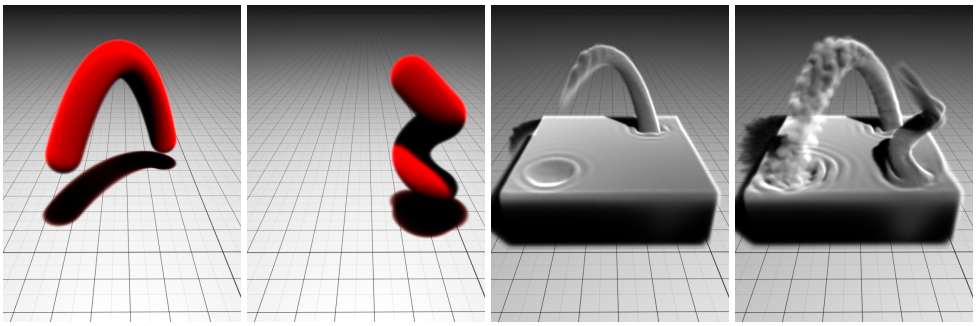


Figure 5 Simulations guided with a user-created flow induced in a narrow tube along the curves shown in red. The guiding weights are animated in order to first enable the leftmost curve-flow and then later the rightmost. **Left pair:** The narrow tubes. **Right pair:** Our method guiding with the animated guiding weights combined with the eroded densities and the error estimate.

creating artistic effects.

ACKNOWLEDGEMENTS

This work was partially funded by the Danish Agency for Science, Technology and Innovation. We wish to thank Peter Trier for acting as a consultant on rendering issues (see http://cg.alexandra.dk/), and Jesper Mosegaard, Thomas Sangild, Clemens Nylandsted Klokmose, Ole Østerby and the anonymous reviewers for their comments. We are grateful to Ken Museth, Doug Roble, Nafees Bin Zafar, Michael Clive and Ryo Sakaguchi for inspiring discussions.

A FULL DERIVATION OF THE GUIDING EQUATIONS

We show here how to derive the guiding equations Eq. (III.5)-Eq. (III.6) from the constrained minimization problem (III.1) – (III.4). We focus on obtaining Eq. (III.5) from (III.1) – (III.4). Note that Eq. (III.4) will also contribute to Eq. (III.5) by means of Lagrange multipliers. The minimization terms (III.1) – (III.3) are very similar, and we concentrate on the most complicated term (III.3) first. Since $\tilde{\mathbf{v}}$ is known, we can rewrite (III.3) as

$$Q(\mathbf{u}(\mathbf{x})) = \int_{\Omega} |[\mathbf{u}(\mathbf{x}) - [\mathscr{F} * \mathbf{u}]](\mathbf{x}) + \mathbf{C}(\mathbf{x})|^2 \, d\mathbf{x} \tag{III.7}$$

where $\mathbf{C}(\mathbf{x}) = -[\tilde{\mathbf{v}} - [\mathscr{F} * \tilde{\mathbf{v}}]](\mathbf{x})$ is known and independent of the velocity field $\mathbf{v}$ we are solving for. Assume that $\mathbf{v}$ is a minimizer of the functional $Q(\mathbf{u}(\mathbf{x}))$ in Eq. (III.7). We then have $Q_{\min} = Q(\mathbf{v}(\mathbf{x}))$. We now employ calculus of variations by considering the first variation of Q: $\delta Q = Q(\mathbf{v}(\mathbf{x}) + \delta\mathbf{v}(\mathbf{x})) - Q_{\min}$ and wish to find the stationary

point corresponding to $\delta Q = 0$. Expanding $Q(\mathbf{v}(\mathbf{x}) + \delta\mathbf{v}(\mathbf{x}))$ gives

$$\begin{aligned} & Q(\mathbf{v}+\delta\mathbf{v}) \qquad\qquad \text{(III.8)} \\ & = \int_\Omega \{(\mathbf{v}+\delta\mathbf{v})\cdot(\mathbf{v}+\delta\mathbf{v}) + [\mathscr{F} * [\mathbf{v}+\delta\mathbf{v}]]\cdot[\mathscr{F} * [\mathbf{v}+\delta\mathbf{v}]] + \\ & \quad \mathbf{C}\cdot\mathbf{C} - 2(\mathbf{v}+\delta\mathbf{v})\cdot[\mathscr{F} * [\mathbf{v}+\delta\mathbf{v}]] - \\ & \quad 2[\mathscr{F} * [\mathbf{v}+\delta\mathbf{v}]]\cdot\mathbf{C} + 2(\mathbf{v}+\delta\mathbf{v})\cdot\mathbf{C}\}\, d\mathbf{x} \end{aligned}$$

Expanding $Q_{\min}$ gives

$$\begin{aligned} Q_{\min} = \int_\Omega & \{\mathbf{v}\cdot\mathbf{v} + [\mathscr{F} * \mathbf{v}]\cdot[\mathscr{F} * \mathbf{v}] + \mathbf{C}\cdot\mathbf{C} - 2[\mathscr{F} * \mathbf{v}]\cdot\mathbf{v} - \\ & 2[\mathscr{F} * \mathbf{v}]\cdot\mathbf{C} + 2\mathbf{v}\cdot\mathbf{C}\}\, d\mathbf{x} \end{aligned}$$

Expanding $Q(\mathbf{v}+\delta\mathbf{v})$ further and omitting higher order terms in $\delta\mathbf{v}$ such as $2\delta\mathbf{v}\cdot[\mathscr{F} * \delta\mathbf{v}]$, $\delta\mathbf{v}\cdot\delta\mathbf{v}$ and $[\mathscr{F} * \delta\mathbf{v}]\cdot[\mathscr{F} * \delta\mathbf{v}]$, for $\delta Q = Q(\mathbf{v}+\delta\mathbf{v}) - Q_{\min}$ we obtain (to first order in $\delta\mathbf{v}$)

$$\delta Q = 2\int_\Omega \{\delta\mathbf{v}\cdot(\mathbf{v} + \mathbf{C} - [\mathscr{F} * \mathbf{v}])\} + \qquad \text{(III.9)}$$

$$\{[\mathscr{F} * \delta\mathbf{v}]([\mathscr{F} * \mathbf{v}] - \mathbf{v} - \mathbf{C})\}\, d\mathbf{x} \qquad \text{(III.10)}$$

To obtain a sufficient condition for a stationary point it is convenient to rewrite δQ in the form $\delta Q = \int_\Omega \delta\mathbf{v}(\mathbf{x})\cdot\mathbf{D}(\mathbf{x})d\mathbf{x}$, where $\mathbf{D}$ is some vector expression. As can be seen, (III.9) is already in this form, and we now focus on rewriting (III.10) in this form as well. By utilizing the definition of the convolution operator, exchanging integration orders and introducing the integration variable $\mathbf{y}$ which ranges over all positions in the domain, we obtain from (III.9) and (III.10)

$$\begin{aligned} \delta Q = 2\int_\Omega \delta\mathbf{v}\cdot\Big\{ & \mathbf{v}(\mathbf{x}) + \mathbf{C}(\mathbf{x}) - [\mathscr{F} * \mathbf{v}](\mathbf{x}) + \\ & \int_\Omega \mathscr{F}(\mathbf{y}-\mathbf{x})([\mathscr{F} * \mathbf{v}](\mathbf{y}) - \mathbf{v}(\mathbf{y}) - \mathbf{C}(\mathbf{y}))\, d\mathbf{y}\Big\} d\mathbf{x} \end{aligned}$$

Since $\delta\mathbf{v}$ is arbitrary, a sufficient condition for $\delta Q = 0$ is that the expression within the curly braces is identically zero. This completes the derivation of the contribution to the Euler-Lagrange equations from (III.3). The contributions from (III.1) and (III.2) are derived similarly. Furthermore, the derivation of the contribution, $\nabla p(\mathbf{x})$, from Eq. (III.4) by the method of Lagrange multipliers is similar to the derivation in [29, p.203]. The contributions from (III.1) – (III.4) are finally added together and set equal to zero

$$\int_\Omega [1-\phi(\mathbf{y})]\mathscr{F}(\mathbf{y}-\mathbf{x})([\mathscr{F} * \mathbf{v}](\mathbf{y}) - [\mathscr{F} * \tilde{\mathbf{v}}](\mathbf{y}))\, d\mathbf{y} \qquad \text{(III.11)}$$

$$+ \int_\Omega \phi(\mathbf{y})\mathscr{F}(\mathbf{y}-\mathbf{x})([\mathscr{F} * \mathbf{v}](\mathbf{y}) - \mathbf{v}_\mathbf{g}(\mathbf{y}))\, d\mathbf{y} \qquad \text{(III.12)}$$

$$+ \mathbf{v}(\mathbf{x}) + \mathbf{C}(\mathbf{x}) - [\mathscr{F} * \mathbf{v}](\mathbf{x}) \qquad \text{(III.13)}$$

$$+ \int_\Omega \mathscr{F}(\mathbf{y}-\mathbf{x})([\mathscr{F} * \mathbf{v}](\mathbf{y}) - \mathbf{v}(\mathbf{y}) - \mathbf{C}(\mathbf{y}))\, d\mathbf{y} \qquad \text{(III.14)}$$

$$+ \nabla p(\mathbf{x}) \qquad \text{(III.15)}$$

$$= 0$$

Expression (III.11) is the contribution from (III.1), (III.12) is the contribution from (III.2), (III.13) – (III.14) are the contributions from (III.3) and finally (III.15) is the contribution from Eq. (III.4). Note that p is the pressure of the fluid. By re-arranging these equations and moving the known quantities to the right-hand-side we obtain Eq. (III.5). This completes the derivation.

BIBLIOGRAPHY

[1] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *J. Comput. Phys.*, 118(2):269–277, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1098. 32, 86

[2] A. Aggarwal and S. V. Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/48529.48535. 41, 42, 44, 45, 87

[3] C. Allen, J. M. Cohen, D. Bloom, D. P. Ferreira, S. Hasegawa, and C. McMahon. Levelsets in production: Spider-man 3. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 29, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1278780.1278815. 2, 15

[4] P. Angot, C.-H. Bruneau, and P. Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik*, 81(4):497–520, February 1999. doi: http://dx.doi.org/10.1007/s002110050401. 69, 133, 134, 149

[5] J. A. Baerentzen and H. Aanaes. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, 2005. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2005.49. 18, 20

[6] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*, 26(3):100, 2007. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1276377.1276502. 62, 141

[7] M. Bergou, S. Mathur, M. Wardetzky, and E. Grinspun. Tracks: toward directable thin shells. *ACM Trans. Graph.*, 26(3):50, 2007. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1276377.1276439. 126, 144

[8] M. Bertalmío, L.-T. Cheng, S. Osher, and G. Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174(2):759–780, 2001. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.2001.6937. 86

[9] A. Bibireata, S. Krishnan, G. Baumgartner, D. Cociorva, C. Lam, P. Sadayappan, J. Ramanujam, D. Bernholdt, and V. Choppella. Memory-constrained data

locality optimization for tensor contractions. In L. Rauchwerger, editor, *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC '03)*, volume 2958 of *Lecture Notes in Computer Science*, pages 93–108. Springer-Verlag, 2004. 88

[10] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Trans. Vis. Comput. Graph.*, 7(2):173–192, 2001. 15, 17

[11] R. Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003. 3, 35, 36, 38, 86

[12] R. Bridson. *Fluid Simulation for Computer Graphics*. AK Peters, 2008. 6, 49, 50, 52, 57, 60, 62, 138

[13] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Computer Animation*. ACM, The Eurographics Association, 2003. 20

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2000. ISBN 0-89871-462-1. 64, 70, 71, 72, 133, 136, 149

[15] S. Chen, G. D. Doolen, and K. G. Eggert. Lattice-boltzmann fluid dynamics. *Los Alamos Science*, (22):98–111, 1994. 58, 78

[16] D. L. Chopp. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106:77–91, 1993. 32, 86

[17] A. Dervieux and F. Thomasset. A finite element method for the simulation of raleigh-taylor instability. *Springer Lect. Notes in Math.*, 771:145–158, 1979. 85

[18] A. Dervieux and F. Thomasset. Multifluid incompressible flows by a finite element method. In W. Reynolds and R. MacCormack, editors, *Seventh International Conference on Numerical Methods in Fluid Dynamics*, volume 141 of *Lecture Notes in Physics*, pages 158–163, 1980. 85

[19] M. Desbrun and M.-P. Gascuel. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 61–76, New York, NY, USA, 1996. Springer-Verlag New York, Inc. ISBN 3-211-82885-0. 53, 61

[20] M. Droske and M. Rumpf. A level set formulation for willmore flow. *Interfaces and Free Boundaries*, 6(3):361–378, 2004. 86

[21] T. F. Dupont and Y. Liu. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *J. Comput. Phys.*, 190(1):311–324, 2003. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/S0021-9991(03)00276-6. 61, 87, 119

[22] T. F. Dupont and Y. Liu. Back and forth error compensation and correction methods for semi-lagrangian schemes with application to level set interface computations. *Mathematics of Computation*, 76:647–668, 2007. 149

[23] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, 2002. 37, 112

[24] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of SIGGRAPH 2002*, Computer Graphics Proceedings, Annual Conference Series, pages 736–744. ACM, ACM Press / ACM SIGGRAPH, 2002. 15, 16, 20, 65, 79

[25] R. Fattal and D. Lischinski. Target-driven smoke animation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 441–448, New York, NY, USA, 2004. ACM. doi: http://doi.acm.org/10.1145/1186562.1015743. 66, 73, 79, 146

[26] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22, Aug. 2001. 57, 60, 62, 126, 132, 137, 141, 144, 149

[27] B. Feldman, J. O'Brien, and O. Arikan. Animating suspended particle explosions, 2003. ISSN 0730-0301. 65

[28] B. E. Feldman, J. F. O'Brien, B. M. Klingner, and T. G. Goktekin. Fluids in deforming meshes. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 255–259, New York, NY, USA, 2005. ACM. ISBN 1-7695-2270-X. doi: http://doi.acm.org/10.1145/1073368.1073405. 62

[29] J. H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 2002. ISBN 3-540-42074-6. 68, 129, 131, 147, 148, 155

[30] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 23–30. ACM SIGGRAPH, ACM Press / ACM SIGGRAPH, Aug. 2001. 2, 15, 16, 57, 86

[31] N. Foster and D. Metaxas. Realistic animation of liquids. In *Graphics Interface '96*, pages 204–212, May 1996. 55, 57

[32] N. Foster and D. Metaxas. Controlling fluid animation. In *CGI '97: Proceedings of the 1997 Conference on Computer Graphics International*, pages 178–188, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7825-9. 57, 127, 144, 146

[33] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, New York, New York, Oct. 17–19 1999. 41, 42, 44, 45, 87

[34] S. F. Frisken and R. Perry. Simple and efficient traversal methods for quadtrees and octrees. *journal of graphics, gpu, and game tools*, 7(3):1–11, 2002. 35

[35] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 249–254. ACM, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 34

[36] E. Froemling, T. Goktekin, and D. Peachey. Simulating whitewater rapids in ratatouille. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 68, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1278780.1278862. 3, 63, 126

[37] F. Gibou, R. Fedkiw, R. Caflisch, and S. Osher. A level set approach for the numerical simulation of dendritic growth. *J. Sci. Comput.*, 19(1-3):183–199, 2003. ISSN 0885-7474. 86

[38] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, September 2001. ISBN 0471383651. 41

[39] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. Graph.*, 22(3):871–878, 2003. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/882262.882358. 20

[40] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 973–981, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1186822.1073299. 62, 141

[41] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965. doi: 10.1063/1.1761178. 55, 61

[42] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *J. Comput. Phys.*, 131(1):3–47, 1997. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1996.5632. 87

[43] S. E. Hieber and P. Koumoutsakos. A lagrangian particle level set method. *J. Comput. Phys.*, 210(1):342–367, 2005. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2005.04.013. 86

[44] J.-M. Hong and C.-H. Kim. Controlling fluid animation with geometric potential: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):147–157, 2004. ISSN 1546-4261. doi: http://dx.doi.org/10.1002/cav.v15:3/4. 146

[45] J.-M. Hong and C.-H. Kim. Discontinuous fluids. *ACM Trans. Graph.*, 24(3):915–920, 2005. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1073204.1073283. 52

[46] B. Houston, M. Wiebe, and C. Batty. RLE sparse level sets. In *Proceedings of the SIGGRAPH 2004 Conference on Sketches & Applications*. ACM, ACM Press, 2004. 36

[47] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation. *ACM Transactions on Graphics*, 25(1):1–24, 2006. 37, 86, 112

[48] U.-L. P. Hy Trac. Out-of-core hydrodynamic simulations for cosmological applications. *New Astronomy*, 2006. 88

[49] W.-K. Jeong and R. T. Whitaker. A fast iterative method for eikonal equations. *SIAM J. Sci. Comput.*, 30(5):2512–2534, 2008. ISSN 1064-8275. doi: http://dx.doi.org/10.1137/060670298. 28, 45, 87, 98

[50] G.-S. Jiang and D. Peng. Weighted eno schemes for hamilton–jacobi equations. *SIAM J. Sci. Comput.*, 21(6):2126–2143, 1999. ISSN 1064-8275. doi: http://dx.doi.org/10.1137/S106482759732455X. 87

[51] G.-S. Jiang and D. Peng. Weighted ENO schemes for Hamilton-Jacobi equations. *SIAM J. Sci. Comput.*, 21(6):2126–2143, 1999. ISSN 1064-8275. 24, 35, 119

[52] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted eno schemes. *J. Comput. Phys.*, 126(1):202–228, 1996. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1996.0130. 24, 35, 87, 119

[53] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Implicit and explicit optimizations for stencil computations. In *MSPC '06: Proceedings of the 2006 workshop on Memory system performance and correctness*, pages 51–60, New York, NY, USA, 2006. ACM. ISBN 1-59593-578-9. doi: http://doi.acm.org/10.1145/1178597.1178605. 89

[54] M. Kandemir, A. Choudhary, J. Ramanujam, and M. A. Kandaswamy. A unified framework for optimizing locality, parallelism, and communication in out-of-core computations. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):648–668, 2000. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/71.877759. 89

[55] M. Kandemir, A. Choudhary, and J. Ramanujam. An i/o-conscious tiling strategy for disk-resident data sets. *J. Supercomput.*, 21(3):257–284, 2002. ISSN 0920-8542. doi: http://dx.doi.org/10.1023/A:1014156327748. 89

[56] R. Keys. Cubic convolution interpolation for digital image processing. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 29(6):1153–1160, 1981. 136

[57] B. Kim, Y. Liu, I. Llamas, and J. Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):135–144, 2007. ISSN 1077-2626. doi: http://dx.doi.org/10.1109/TVCG.2007.3. 137

[58] T. Kim, N. Thürey, D. James, and M. Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):1–6, 2008. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1360612.1360649. 66, 69, 73, 127, 128, 146, 153

[59] Y. Kim, R. Machiraju, and D. Thompson. Path-based control of smoke simulations. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 33–42, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN 3-905673-34-7. 146

[60] D. Kincaid and W. Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. Brooks/Cole, Pacific Grove, CA, USA, third edition, 2002. ISBN 0-534-38905-8. 23, 24, 53, 55, 57, 60

[61] P. Koumoutsakos, G.-H. Cottet, and D. Rossinelli. Flow simulations using particles: bridging computer graphics and cfd. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–73, New York, NY, USA, 2008. ACM. doi: http://doi.acm.org/10.1145/1401132.1401166. 133, 134, 149

[62] M. Kowarschik. An overview of cache optimization techniques and cache-aware numerical algorithms. In *In Algorithms for Memory Hierarchies, volume 2625 of LNCS*, pages 213–232. Springer, 2003. 89

[63] C. Lanczos. *The Variational Principles of Mechanics*. Dover Publications, fourth edition, 1986. 8, 64, 68

[64] A. E. Lefohn, J. E. Cates, and R. T. Whitaker. Interactive, gpu-based level sets for 3d segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003*, pages 564–572, 2003. ISBN 978-3-540-20462-6. 15, 32

[65] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 11, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2030-8. doi: http://dx.doi.org/10.1109/VISUAL.2003.1250357. 86

[66] A. Leventhal. Flash storage memory. *Commun. ACM*, 51(7):47–51, 2008. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1364782.1364796. 86

[67] R. LeVeque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal.*, 33:627–665, 1996. 112

[68] Z. Li and Y. Song. Automatic tiling of iterative stencil loops. *ACM Trans. Program. Lang. Syst.*, 26(6):975–1028, 2004. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/1034774.1034777. 94

[69] X. Liu, S. Osher, and T. Chan. Weighted essentially nonoscillatory schemes. *J. Comput. Phys.*, 115:200–212, 1994. 24, 35, 61, 119

[70] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *J. Comput. Phys.*, 115(1):200–212, 1994. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1994.1187. 87

[71] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual*

conference on Computer graphics and interactive techniques, pages 163–169, New York, NY, USA, 1987. ACM Press. ISBN 0-89791-227-6. doi: http://doi.acm.org/10.1145/37401.37422. 5, 113

[72] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3), Aug. 2004. 3, 34, 35

[73] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids (in review)*, 2005. 86

[74] K. S. Mckinley and S. Carr. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18:424–453, 1996. 89

[75] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 449–456, New York, NY, USA, 2004. ACM. doi: http://doi.acm.org/10.1145/1186562.1015744. 65, 127, 146

[76] R. B. Milne. *An Adaptive Level-Set Method*. PhD thesis, University of California, Berkeley, 1995. 86

[77] C. Min. Local level set method in high dimension and codimension. *Journal of Computational Physics*, 200:368–382, 2004. 35, 86

[78] J. Molemaker, J. M. Cohen, S. Patel, and J. Noh. Low viscosity flow simulations for animation. In *ACM SIGGRAPH Symposium on Computer Animation*, 2008. 134

[79] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun. Energy-preserving integrators for fluid animation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–8, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-726-4. doi: http://doi.acm.org/10.1145/1576246.1531344. 128, 144, 146

[80] M. Müller, D. Charypar, and M. 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. Eurographics Association, 2003. ISBN 1-58113-659-5. 53, 61

[81] K. Museth. An efficient level set toolkit for visual effects. In *SIGGRAPH '09: SIGGRAPH 2009: Talks*, pages 1–1, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-834-6. doi: http://doi.acm.org/10.1145/1597990.1597995. 37, 86

[82] K. Museth and M. Clive. Cracktastic: fast 3d fragmentation in "the mummy: Tomb of the dragon emperor". In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks*, pages 1–1, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-343-3. doi: http://doi.acm.org/10.1145/1401032.1401110. 37, 86

[83] K. Museth, D. Breen, R. Whitaker, and A. Barr. Level set surface editing operators. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 21(3):330–338, July 2002. 15, 16, 86

[84] K. Museth, D. Breen, R. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24(4): 821–841, 2005. 19

[85] K. Museth, M. Clive, and N. B. Zafar. Blobtacular: surfacing particle system in "pirates of the caribbean 3". In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 20, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1278780.1278804. 37

[86] R. Narain, J. Sewall, M. Carlson, and M. C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM. doi: http://doi.acm.org/10.1145/1457515.1409119. 127, 128, 146

[87] O. Nemitz, M. B. Nielsen, M. Rumpf, and R. Whitaker. Finite element methods on very large, dynamic tubular grid encoded implicit surfaces. *SIAM Journal on Scientific Computing*, 31(3):2258–2281, 2009. doi: 10.1137/080718334. 86

[88] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 721–728, New York, NY, USA, 2002. ACM. ISBN 1-58113-521-1. doi: http://doi.acm.org/10.1145/566570.566643. 2, 15, 16

[89] M. B. Nielsen. *Efficient and High Resolution Level Set Simulations*. PhD thesis, Aarhus University, 2006. 90, 105

[90] M. B. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput.*, 26(3):261–299, 2006. ISSN 0885-7474. doi: http://dx.doi.org/10.1007/s10915-005-9062-8. 3, 5, 36, 45, 86, 88, 99, 100, 106

[91] M. B. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set methods. *ACM Trans. Graph.*, 26(4):26, 2007. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1289603.1289607. 7, 38, 39, 40, 41, 44, 45, 46, 80, 86, 87, 88, 90, 91, 92, 106, 107, 108, 109

[92] M. B. Nielsen, B. B. Christensen, N. B. Zafar, D. Roble, and K. Museth. Guiding of smoke animations through variational coupling of simulations at different resolution. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 206–215, Aug. 2009. 144, 145, 146, 147, 148, 149, 151, 153

[93] O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis: an eulerian approach with lagrangian particle tracking. In *Visualization, 2005. VIS 05. IEEE*, pages 407–414, Oct. 2005. doi: 10.1109/VISUAL.2005.1532823. 32, 33, 34

[94] S. Osher and N. Paragios. *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. ISBN 0387954880. 15

[95] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988. 2, 3, 15, 20, 31, 85

[96] S. Osher and C. Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Num. Anal.*, 28:907–922, 1991. 24, 34

[97] S. Osher, L.-T. Cheng, M. Kang, H. Shim, and Y.-H. Tsai. Geometric optics in a phase-space-based level set and eulerian framework. *J. Comput. Phys.*, 179 (2):622–648, 2002. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.2002.7080. 86

[98] S. J. Osher and R. P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, October 2002. ISBN 0387954821. 19, 21, 22, 25, 26, 27, 29

[99] O. Østerby. *Numerical Solution of Parabolic Equations*. DAIMI FN. Second edition, January 2008. 25, 29

[100] P. Oswald. Remarks on multilevel bases for divergence-free finite elements. *Numerical Algorithms*, 2001. 142

[101] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-based fast local level set method. *J. Comput. Phys.*, 155(2):410–438, 1999. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1999.6345. 3, 22, 27, 32, 33, 34, 86

[102] T. Pfaff, N. Thuerey, A. Selle, and M. Gross. Synthetic turbulence using artificial boundary layers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, New York, NY, USA, 2009. ACM. 146

[103] J. Pilgrim and T. Tornberg. How to build a sixty foot man of moving sand. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 28, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1278780.1278814. 2, 15

[104] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, first edition, 2007. 109, 110, 137

[105] R. Rickitt. *Special Effects: The History and Technique*. Aurum Press Ltd, London, UK, 2006. ISBN 1845131304. 48, 49

[106] J. K. Salmon and M. S. Warren. Parallel, out-of-core methods for n-body simulation. In *PPSC*, 1997. 88

[107] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, 2008. 127, 128, 146

[108] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, 2005. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1073204.1073282. 60

[109] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. of the National Academy of Sciences of the USA*, 93(4):1591–1595, February 1996. 27, 98

[110] L. Shi and Y. 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. ISBN 1-7695-2270-X. doi: http://doi.acm.org/10.1145/1073368.1073401. 128, 146

[111] P. Shirley and R. K. Morley. *Realistic Ray Tracing*. A. K. Peters, Ltd., Natick, MA, USA, 2003. ISBN 1568811985. 5

[112] C. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes. *J. Comput. Phys.*, 77:439–471, 1988. 25, 61, 87, 94, 119

[113] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 215–228, 1999. 89

[114] J. Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 121–128, Aug. 1999. 5, 53, 59, 63, 126, 128, 144, 145

[115] Stanford Scanning Repository. Stanford scanning repository. http://graphics.stanford.edu/data/3Dscanrep/. 112

[116] J. Steinhoff and D. Underhill. Modification of the euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6(8):2738–2744, 1994. doi: 10.1063/1.868164. 60

[117] J. Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, 151(2):616–648, 1999. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1999.6205. 34, 35, 86

[118] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114(1):146–159, 1994. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1994.1155. 22, 85

[119] M. Sussman, A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. An adaptive level set approach for incompressible two-phase flows. *J. Comput. Phys.*, 148(1):81–124, 1999. ISSN 0021-9991. 86

[120] A. S. Tanenbaum and J. R. Goodman. *Structured Computer Organization*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. ISBN 0130959901. 41, 42

[121] N. Thürey and U. Rüde. Free Surface Lattice-Boltzmann fluid simulations with and without level sets. *Proc. of Vision, Modelling, and Visualization VMV*, pages 199–207, 2004. 58

[122] N. Thürey, K. Iglberger, and U. Rüde. Free Surface Flows with Moving and Deforming Objects for LBM. *Proceedings of Vision, Modeling and Visualization 2006*, pages 193–200, Nov 2006. 58

[123] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 7–12, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN 3-905673-34-7. 73, 79, 126, 127, 128, 138, 144, 146, 153

[124] N. Thürey, T. Pohl, and U. Rüde. Hybrid Parallelization Techniques for Lattice Boltzmann Free Surface Flows. *Proceedings of Parallel CFD 2007*, pages 1–8, 2007. 58

[125] S. Toledo. A survey of out-of-core algorithms in numerical linear algebra. pages 161–179, 1999. 40, 86, 88

[126] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 716–723, New York, NY, USA, 2003. ACM. ISBN 1-58113-709-5. doi: http://doi.acm.org/10.1145/1201775.882337. 65, 127, 146

[127] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, LF*, pages 1368–1373, December 1994. 28, 98

[128] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. Automat. Contr.*, 40:1528–1538, September 1995. 28, 98

[129] C. D. Twigg and D. L. James. Many-worlds browsing for control of multibody dynamics. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), Aug. 2007. 79

[130] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Comput. Surv.*, 33(2):209–271, 2001. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/384192.384193. 40, 86, 88

[131] R. Whitaker, D. Breen, K. Museth, and N. Soni. A framework for level set segmentation of volume datasets. In *Proceedings of International Workshop on Volume Graphics*, pages 159–168, 2001. 15

[132] R. T. Whitaker. A level-set approach to 3d reconstruction from range data. *Int. J. Comput. Vision*, 29(3):203–231, 1998. ISSN 0920-5691. doi: http://dx.doi.org/10.1023/A:1008036829907. 32, 86

[133] M. Wiebe and B. Houston. The Tar Monster: Creating a character with fluid simulation. In *Proceedings of the SIGGRAPH 2004 Conference on Sketches & Applications*. ACM, ACM Press, 2004. 4

[134] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *PLDI '91: Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 30–44, New York, NY, USA, 1991. ACM Press. ISBN 0-89791-428-7. doi: http://doi.acm.org/10.1145/113445.113449. 42, 44, 45, 87, 89, 94, 95, 116, 117

[135] D. Wonnacott. Achieving scalable locality with time skewing. *Int. J. Parallel Program.*, 30(3):181–221, 2002. ISSN 0885-7458. doi: http://dx.doi.org/10.1023/A:1015460304860. 89, 92, 94, 95, 96

[136] Y. Zhu and R. Bridson. Animating sand as a fluid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 965–972, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1186822.1073298. 61