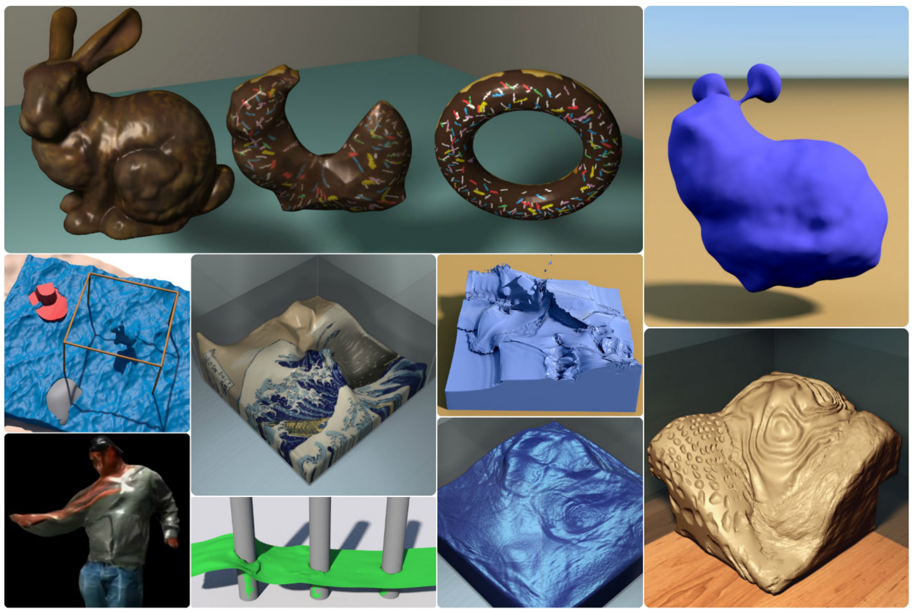
Tracking, Correcting and Absorbing Water Surface Waves



Thesis by

Morten Bojsen-Hansen

(Defended July 15, 2016)

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Institute of Science and Technology Austria

Klosterneuburg, Austria

Approval

The thesis by

Morten Bojsen-Hansen

titled

Tracking, Correcting and Absorbing Water Surface Waves

is approved by:

Supervisor: Chris Wojtan, *IST Austria*

Signature: [illegible]

Committee member: Hao Li, *University of Southern California*

Signature: HL

Committee member: Vladimir Kolmogorov, *IST Austria*

Signature: [illegible]

Committee member: Nils Thürey, *TU Munich*

Signature: [illegible]

Defense chair: Ryuichi Shigemoto, *IST Austria*

Signature: 重本陽一

Declaration

I hereby declare that this dissertation is my own work, and it does not contain other peoples work without this being so stated; and this thesis does not contain my previous work without this being stated, and that the bibliography contains all the literature that I used in writing the dissertation, and that all references refer to this bibliography.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other University or Institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

[illegible]

Morten Bojsen-Hansen
Klosterneuburg, Austria
June 5, 2016

Abstract

Computer graphics is an extremely exciting field for two reasons. On the one hand, there is a healthy injection of pragmatism coming from the visual effects industry that want robust algorithms that work so they can produce results at an increasingly frantic pace. On the other hand, they must always try to push the envelope and achieve the impossible to wow their audiences in the next blockbuster, which means that the industry has not succumb to conservatism, and there is *plenty* of room to try out new and *crazy* ideas if there is a chance that it will pan into something useful.

Water simulation has been in visual effects for decades, however it still remains extremely challenging because of its high computational cost and difficult art-directability. The work in this thesis tries to address some of these difficulties. Specifically, we make the following three novel contributions to the state-of-the-art in water simulation for visual effects.

First, we develop the first algorithm that can convert *any* sequence of closed surfaces in time into a moving triangle mesh. State-of-the-art methods at the time could only handle surfaces with *fixed* connectivity, but we are the first to be able to handle surfaces that merge and split apart. This is important for water simulation practitioners, because it allows them to convert splashy water surfaces extracted from particles or simulated using grid-based level sets into triangle meshes that can be either textured and enhanced with extra surface dynamics as a post-process. We also apply our algorithm to other phenomena that merge and split apart, such as morphs and noisy reconstructions of human performances.

Second, we formulate a surface-based energy that measures the deviation of a water surface from a physically valid state. Such discrepancies arise when there is a mismatch in the degrees of freedom between the water surface and the underlying physics solver. This commonly happens when practitioners use a moving triangle mesh with a grid-based physics solver, or when high-resolution grid-based surfaces are combined with low-resolution physics. Following the direction of steepest descent on our surface-based energy, we can either smooth these artifacts or turn them into high-resolution waves by interpreting the energy as a physical potential.

Third, we extend state-of-the-art techniques in non-reflecting boundaries to handle spatially and time-varying background flows. This allows a novel new workflow where practitioners can re-simulate part of an existing simulation, such as removing a solid obstacle, adding a new splash or locally changing the resolution. Such changes can easily lead to new waves in the re-simulated region that would reflect off of the new simulation boundary, effectively ruining the illusion of a seamless simulation boundary between the existing and new simulations. Our non-reflecting boundaries makes sure that such waves are absorbed.

Acknowledgments

First and foremost I would like to thank **Chris**. I have been incredibly lucky to have you as my advisor. Your integrity and aspiration to do *the right thing* in all walks of life is something I admire and aspire to. I also really appreciate the fact that when working with you it felt like we were equals. I think we had a very synergetic work relationship: I learned immensely from you, but I dare say that you learned a few things from me as well. ;)

Next, I would like to thank my amazing committee. **Hao**, it was a fantastic experience working with you. You showed me how to persevere and keep morale high when things were looking the most bleak before the deadline. You are an incredible motivator and super fun to be around! **Vladimir**, thanks for the shared lunches and the poker games. Sorry for not bringing them back when I got busy. Also, sorry for embarrassing you by asking about your guitar playing that one time. You really are quite awesome! **Nils**, one of the friendliest and most humble people you will meet and a top notch researcher to boot! Thank you for joining my committee late!

I would also like to acknowledge the Visual Computing group at IST Austria from whom I have learned so much. The excellent discussions we had in reading groups and research meetings really helped me become a better researcher!

Next, I would like to thank all the amazing people that I met during my PhD studies, both at IST Austria, in Vienna and elsewhere. I have mentioned this many times before, but coming to Austria has been the best decision of my life so far. I am a totally different person now from when I left Denmark, and I chalk that up to the positive influence of all the people that I had the greatest fortune of meeting and interacting with. Everyone has helped shape my life in one way or another. I think you all know who you are. :)

Lastly, I would like to thank my family, my parents and my two brothers, for putting up with me throughout all the years. Sometimes I can be a bit uncompromising when I pursue my lofty goals, but it really means everything to me that I can count on you. I could not have done this without your loving support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Visual effects has long been a staple in the advertisement and entertainment industries, essential to achieving physically plausible yet artistically directable animations. The use of simulated fluids for such purposes is already a rather mature enterprise and most effects can be achieved through use of general purpose algorithms and a lots of laborious work by an army of skilled artists to polish the output of said algorithms. However, audiences are always looking to the next blockbuster for ever more impressive computer-generated imagery and budgets keep spiralling upwards.

Simply scaling up the resolution is seldom a solution because one quickly exceeds computational budgets due to the super-volumetric space and time complexity of commonly-used simulation algorithms. Although many parts of these algorithms are embarrassingly parallel and thus benefit from Moore's law, there are many indicators that this exponential explosion in computational power is coming to an end. Even in the best case where Moore's law carries on indefinitely, the quality that can be achieved by simply turning the resolution knob also has a point of diminishing returns. Setting aside the issue of limited computation time, there are also serious issues inherit in the traditional and somewhat laborious workflow of continuously tweaking initial conditions and waiting for and inspecting the simulation result, chiefly in terms of costly artist time.

As existing techniques have matured, the focus in research has broadened considerably and attention has shifted away from more traditional techniques and data structures like finite differences and adaptive grids that could equally be applied in computation fluid dynamics, to more computer graphics specific ideas, such as alternative data structures [Wojtan et al. 2009; Brochu et al. 2010; Da et al. 2014] and integration methods [Elcott et al. 2007; Mullen et al. 2009; De Witt et al. 2012; Ando et al. 2015; Da et al. 2016], art-direction [McNamara et al. 2004; Raveendran et al. 2012; Pan et al. 2013], re-using existing simulation data [Treuille et al. 2006; Nielsen and Bridson 2011; Raveendran et al. 2014], and reproducing specific natural phenomena [Tessendorf 2004; Kim et al. 2010; Zhu et al. 2015; Chern et al. 2016] and various physically-inspired approximations [Goktekin et al. 2004; Kim et al. 2009; Thürey et al. 2010].

The work presented in this thesis intends to contribute to this general trend in simulation techniques for liquids with methods that empower artists with new possibilities, more efficient algorithms and more control. In the next few sections we motivate several avenues for such improvements as well as briefly summarize our proposed solutions and contributions to the field of computer graphics.

1.1 Tracking liquid surface waves

An essential part of any liquid simulation algorithms is a representation of the liquid interface. Traditionally, practitioners have preferred methods that do not explicitly maintain the points on the liquid surface such as particles [Zhu and Bridson 2005] or level sets [Osher and Fedkiw 2003] (see Section 2.10.2) because such representations automatically handle splitting and merging events such as when droplets pinch off or a wave overturns. Unfortunately, these methods do not come equipped with a natural surface parameterization, which prevents surface-based enhancement techniques like texturing or surface-based dynamics useful for enriching the simulation results in post-production.

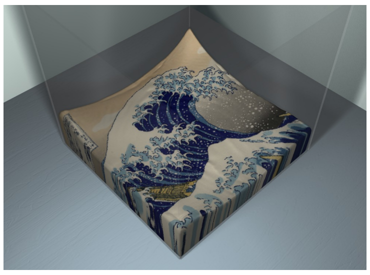


Figure 1.1: Our tracking algorithm applied to a liquid simulation. We applied the image of The Great Wave off Kanagawa at the last frame and propagated it backwards to the first frame.

Conversely, moving triangle meshes explicitly maintain the surface, which allows easy and efficient implementation of surface-based algorithms at the cost that splitting and merging events now have to be handled explicitly. Perhaps owing to this difficulty, robust algorithms for moving triangle meshes only appeared recently, and have not yet been widely adopted. Naturally, there is a great deal of inertia associated with existing particle and level set pipelines, and it will take considerable time before tooling for moving triangle meshes catches up.

In Chapter 3 we help alleviate this problem. We present a fully automatic method that converts *any* surface representation, including level sets and surfaces extracted from particles, into a temporally coherent moving triangle mesh. This allows practitioners to take advantage of all the great benefits that you get from moving triangle meshes without replacing existing pipelines. It also allows moving triangle meshes, which are more susceptible to numerical problems due to collision detection and mesh surgery, to mature and become more robust while they happily coexist alongside the existing pipelines. The only input to our algorithm is a sequence of input surfaces, so we are not restricted to liquid simulations. In our examples we apply our algorithm to physics-based simulations, artistic morphs and human performance capture data acquired from the real world. This allows us to add displacement maps to a simulation of goo, morph a chocolate-textured bunny into a donut and simulate surface waves on top of a level set-based liquid simulation.

1.2 Correcting liquid surface waves

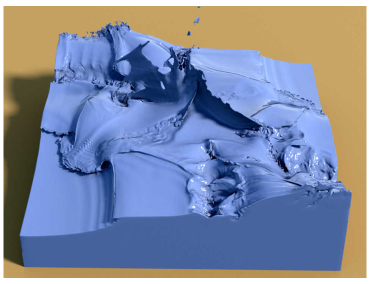


Figure 1.2: Our surface correction applied to liquid simulation with artifacts. The articats are turned into gravity waves, which give the illusion of a much higher resolution than the underlying grid-based physics.

There are two major components in any liquid simulator. First, there is a physics solver that evolves the volumetric velocity by numerically integrating the Navier-Stokes equations forward in time. Second, is the surface tracker which moves the interface that delineates the boundary between the liquid and its surrounding environment, typically the air and various solids. Since the velocity field is invisible, the liquid surface is typically the only thing that is directly observable, so it makes sense to concentrate computational effort here. This point is further exasperated by the fact that the computation time of algorithms for numerically integrating the velocity field scales with the liquid volume, whereas for surface evolution algorithms, it typically scales with surface area.

In order for the surface tracker to remain in a physically valid state at all times, it is imperative that the degrees of freedom of the surface tracker and the physics solver are aligned so that any change in the surface can be matched by a corresponding force from the physics solver. This is not generally possible when combining disparate solvers such as grid-based physics and a moving triangle mesh for tracking the surface, nor is it possible when the surface tracker has more degrees of freedom than the physics.

That being said, practitioners nevertheless like to use a mismatched resolution between the surface tracker and the physics solver because it gives the appearance of more detailed physics at a much more favorable algorithmic complexity as explained above. This leads to inevitable surface artifacts, which are either ignored or partially alleviated with various heuristics that typically lack a physical justification such as *e.g.* smoothing.

In Chapter 4 we construct a simple surface-based energy that characterizes the degree to which the surface deviates from a physically valid state. We may use the gradient of this energy in one of two ways. By following the direction of steepest descent we arrive at a surface smoothing algorithm that aggressively attacks physically inconsistent states but leaves physically valid, and thus desirable, surface details. Interpreting the energy as a physical potential, we may alternatively turn surface artifacts into gravity or surface tension waves, greatly enriching the surface motion while avoiding additional costly volumetric computation. Our formulation is general and applies to any surface tracker. We show examples using both moving triangle meshes and level sets.

1.3 Absorbing liquid surface waves

It is easy to see that the traditional approach, where the whole simulation has to be discarded and simulated anew every time the initial conditions change, can lead to a quite laborious and wasteful workflow for the artists. Not only can it be extremely unintuitive to predict how small changes in fluid or solid geometries affect the simulated result several seconds later, but, as is often the case, it is also very difficult for the artist to avoid messing up other parts of the simulation results that they were actually happy with already. Clearly, it would be desirable if artists could instead build up the simulation incrementally without affecting the parts that have already been finalized.

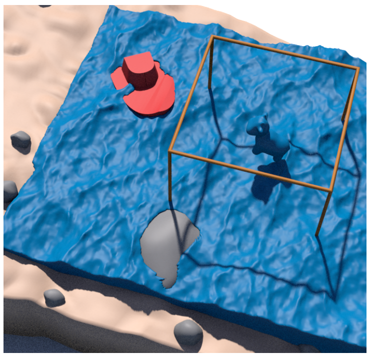


Figure 1.3: We add a splash to an existing liquid simulation. Our method ensures that the radiating waves are absorbed and do not reflect.

One possible approach that has been used in industry is to re-simulate part of an existing simulation by setting the input simulation as a boundary condition, however, this poses problems when waves originating in the new simulation approach the boundary between the new and existing simulations. If no special care is taken, the wave will reflect back into the new simulation and the illusion of a single seamless simulation is destroyed. Naively ramping the velocity towards the input simulation also does not work, since this effectively changes the material properties and thus also causes spurious reflections.

What is needed is an approach that absorbs the emanating waves *without* reflection – a non-reflecting boundary. In Chapter 5 we show how to derive such a boundary layer by extending state-of-the-art perfectly matched layers theory. We show in several examples how our approach can be used to remove solid obstacles, add new splashes and locally increase the resolution without re-doing the whole simulation. As an added benefit, we also achieve healthy computational savings in the common case where the size of the re-simulated domain is much smaller than the input simulation.

1.4 Contributions

The methods presented in this thesis contribute to the computer animation community in several ways. In this section we summarize the most important contributions and defer mention of more detailed and technical contributions to Chapters 3 to 5.

- Previous authors have proposed algorithms that track a rigid or a deforming surface through time. We provide the first fully automatic algorithm that can track surfaces that merge or split apart. Using our algorithm we can convert *any* surface representation, such as level sets or surfaces derived from particles, to a temporally coherent moving triangle mesh. This allows us to texture human performance data containing topological noise and add displacements to simulated blobs of goo, among other things.

- Many authors have proposed surface-based dynamics to enrich existing volumetric liquid simulations with extra surface details. Unfortunately, the waves were usually driven by some measure of curvature simply because there was no other natural events to seed waves in the usual wave equation. This usually lead to very noisy results with the entire surface quickly becoming one big noisy patch with no clear traveling waves. As an alternative, we propose to explicitly detect topological events such as merging and splitting, and seed waves exactly where these events occur. This leads to far less noisy and thus more meaningful and visually satisfying results.

- While many authors have employed a separate decoupled resolution for their surface tracker with various remedies to combat the resulting artifacts, we are first to address the artifacts from first principles in a physically-based manner that can be employed at any scale and not just for *e.g.* surface tension-dominant simulations. The fact that we formulate the error as a surface-based energy allows us to derive both smoothing and force-based algorithms to reduce or eliminate the surface artifacts in a way that is both theoretically and practically superior to previous heuristic methods. The algorithms are also efficient because their computational cost only scales with surface area as opposed to the volumetric simulation they sit on top of.

- We show how a simple derivation of a perturbation relative to an existing fluid simulation can be used to construct the first method for non-reflecting boundaries based on the state-of-the-art theory of perfectly matched layers that supports spatially and time-varying background flows. This enables artists to efficiently re-simulate part of an existing fluid simulation without spurious wave reflections at the simulation boundary, something which was not possible with previous approaches based on perfectly matched layers and only possible without non-reflecting boundaries by using impractically large domains. We show examples of removing solid obstacles, adding a splash and locally increasing the resolution (and thus add more visual detail) as a post-process.

1.5 Publications

As part of the work towards this thesis several scientific articles were published. Our work on tracking time-varying surfaces without any priors about the topology was published as:

> M. Bojsen-Hansen, H. Li, and C. Wojtan. 2012. Tracking Surfaces with Evolving Topology. In *ACM Transactions on Graphics (SIGGRAPH)* 31.4, 53:1–53:10.

Our work about correcting liquid surfaces was published as:

> M. Bojsen-Hansen and C. Wojtan. 2013. Liquid Surface Tracking with Error Compensation. In *ACM Transactions on Graphics (SIGGRAPH)* 32.4, 68:1–68:10.

Finally, our work on absorbing liquid surface waves with applications to liquid re-simulation was published as:

> M. Bojsen-Hansen and C. Wojtan. 2016. Generalized Non-Reflecting Boundaries for Fluid Re-Simulation. In *ACM Transactions on Graphics (SIGGRAPH)* 35.4.

1.6 Outline

The remainder of this thesis is structured as follows. In Chapter 2 I briefly review background material prerequisite for understanding the following chapters. This chapter may be skipped by people already very familiar with fluid simulation in computer graphics. In Chapter 3 we describe our algorithm for tracking time-varying surfaces without any priors on the topology. In Chapter 4 we describe our novel technique for correcting errors in liquid surfaces due to resolution mismatches by either smoothing the errors or turning them into waves. In Chapter 5 we describe how to formulate the dynamics of a perturbation of the Navier-Stokes equations and how to damp this perturbation to zero so we can re-simulate parts of an existing simulation without wave reflections at the boundary. Finally, in Chapter 6 we summarize and conclude the thesis with a brief overview of the research that has been done concurrent to the work described in this thesis.

Chapter 2

Background

In this chapter we briefly summarize the most important background knowledge needed to read the rest of this thesis. The point is to collect, in one place, relevant information otherwise scattered throughout the scientific literature. The exposition is *not* meant to be exhaustive or in-depth, rather we try to get the main point across and leave references for the readers who would like to know more.

2.1 Descent methods

Many problems in computer graphics can be formulated as finding the minimizer of some function that measures the error or deviation from some ideal state. The problems in this thesis are no exception. In Chapter 3 we non-rigidly align one shape with another while minimizing the amount of deformation, and in Chapter 4 we minimize the deviation of a liquid surface from the set of physically valid states. In this section we will briefly review the minimization algorithms employed in this thesis. We refer to the materials that served as inspiration for this section for a more in-depth exposition [Madsen et al. 2004; Pighin and Lewis 2007].

Abstractly, a minimization problem is the task of finding the global minimizer $\mathbf{x}^\dagger \in \mathbb{R}^n$ of a certain problem-specific function $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}$

$$\mathbf{x}^\dagger = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{F}(\mathbf{x}). \tag{2.1}$$

Finding the global minimizer as in Equation (2.1) is generally intractable for non-convex $\mathbf{F}$, so we often settle for a local minimizer $\mathbf{x}^\star \in \mathbb{R}^n$

$$\mathbf{F}(\mathbf{x}^\star) \leq \mathbf{F}(\mathbf{x}) \quad \text{for} \quad ||\mathbf{x}^\star - \mathbf{x}|| < \delta \tag{2.2}$$

for some suitable $\delta > 0$.

The literature on minimization algorithms is extensive and also an active area of research. New application-specific methods that exploit the specific properties of **F** come out all the time. In the following, however, we will focus on a fairly general class of minimization algorithms called *descent methods*.

Descent methods follow a very simple iterative pattern. You start with an initial guess $\mathbf{x}_0$ and then proceed to repeat the following steps repeatedly until convergence, which is determined by $|\mathbf{F}(\mathbf{x}_{n+1}) - \mathbf{F}(\mathbf{x}_n)| < \varepsilon$ for some stopping criterion $\varepsilon > 0$.

1. Choose a descent direction, *i.e.* choose a $\Delta\mathbf{x}$ such that

   $$\mathbf{F}(\mathbf{x}_n + \alpha\Delta\mathbf{x}) < \mathbf{F}(\mathbf{x}_n).$$

   for some $\alpha > 0$.

2. Search for the for the best $\alpha > 0$ such that

   $$\mathbf{F}(\mathbf{x}_n + \alpha\Delta\mathbf{x}) < \mathbf{F}(\mathbf{x}_n + \alpha'\Delta\mathbf{x})$$

   for $\alpha' > 0$ with $|\alpha - \alpha'| < \delta$.

3. Update the current guess, $\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha\Delta\mathbf{x}$.

We will not delve deeper into the details of choosing α in step two since it is in general a nontrivial problem. In the simplest case you can get away with using a small fixed α in each iteration. We refer to the references provided at the beginning of this section for more details on how to perform such *line search*. In the next sections we will look at three common ways of choosing $\Delta\mathbf{x}$ in step one.

2.1.1 Gradient descent

The simplest method of choosing $\Delta\mathbf{x}$ arises from a first-order Taylor approximation of $\mathbf{F}(\mathbf{x} + \Delta\mathbf{x})$ around the current guess $\mathbf{x}$.

$$\begin{aligned}\mathbf{F}(\mathbf{x} + \Delta\mathbf{x}) &= \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x})\Delta\mathbf{x} + O\big(||\Delta\mathbf{x}||^2\big) \\ &= \mathbf{F}(\mathbf{x}) + \cos\theta\,\big\|\mathbf{F}'(\mathbf{x})\big\|\,||\Delta\mathbf{x}|| + O\big(||\Delta\mathbf{x}||^2\big)\end{aligned}$$

where $\mathbf{F}'(\mathbf{x})$ is the Jabobian at $\mathbf{x}$ and θ is the angle between $\mathbf{F}'(\mathbf{x})$ and $\Delta\mathbf{x}$. We see that for sufficiently small $||\Delta\mathbf{x}||$ the fastest way of decreasing $\mathbf{F}$ as a function of $\Delta\mathbf{x}$ is to set

$$\Delta\mathbf{x} = -\mathbf{F}'(x) \tag{2.3}$$

because then $\cos\theta = \cos\pi = -1$. Using the descent direction in Equation (2.3) is called *gradient descent* or *steepest decent* and is the method we use to reduce our energy functional in Chapter 4. Since it is based on a first-order approximation of **F** it converges linearly, which may or may not be fast enough depending on the application or the particular function under consideration.

2.1.2 Newton's method

It is possible to do better than linear convergence if we bump our approximation of $\mathbf{F}(\mathbf{x} + \Delta\mathbf{x})$ to second-order

$$\mathbf{F}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{F}''(\mathbf{x})\Delta\mathbf{x} + O\big(\|\Delta\mathbf{x}\|^3\big).$$

where $\mathbf{F}''$ is the Hessian. Now, let

$$\hat{\mathbf{F}}(\Delta\mathbf{x}) \overset{\text{def}}{=} \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{F}''(\mathbf{x})\Delta\mathbf{x}$$

be the truncated second-order expansion of $\mathbf{F}$ around $\mathbf{x}$ and consider sufficiently small $\Delta\mathbf{x}$ such that $\hat{\mathbf{F}} \approx \mathbf{F}$. We are looking for the $\Delta\mathbf{x}$ that decreases $\hat{\mathbf{F}}$ the fastest. This minimizer can be found by examining the critical points of $\hat{\mathbf{F}}$

$$0 = \frac{\mathrm{d}\hat{\mathbf{F}}(\Delta\mathbf{x})}{\mathrm{d}\Delta\mathbf{x}} = \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\Delta\mathbf{x}$$

and solving for $\Delta\mathbf{x}$

$$\Delta\mathbf{x} = -\big[(\mathbf{F}''(\mathbf{x})\big]^{-1}\mathbf{F}'(\mathbf{x}) \tag{2.4}$$

Using the descent direction in Equation (2.4) is called *Newton's method*. Assuming that the Hessian $\mathbf{F}''$ is positive definite and that we are sufficiently close to $\mathbf{x}^\star$, Newton's method converges quadratically, which may be much faster than the linear convergence of gradient descent. The cost we pay for the improved convergence is that we now have to invert the Hessian.

2.1.3 Gauss-Newton

One downside of Newton's method is that we need second-order derivatives. These can be expensive to compute and numerically problematic depending on the smoothness of $\mathbf{F}$. We can side-step this problem if $\mathbf{F}$ can be written as a sum of squares

$$\mathbf{F}(\mathbf{x}) = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{k} f_i(\mathbf{x})^2 \tag{2.5}$$

In this case it can be shown that

$$\mathbf{F}'(\mathbf{x}) = \mathbf{f}'(\mathbf{x})^{\mathsf{T}}\mathbf{f}(\mathbf{x}) \tag{2.6}$$

$$\mathbf{F}''(\mathbf{x}) = \mathbf{f}'(\mathbf{x})^{\mathsf{T}}\mathbf{f}'(\mathbf{x}) + \mathbf{f}(\mathbf{x})^{\mathsf{T}}\mathbf{f}''(\mathbf{x}) \tag{2.7}$$

If we replace $\mathbf{f}$ in Equation (2.5) with its first-order Taylor expansion

$$\hat{\mathbf{f}}(\Delta\mathbf{x}) \overset{\text{def}}{=} \mathbf{f}(\mathbf{x}) + \mathbf{f}'(\mathbf{x})\Delta\mathbf{x}$$

we instead get

$$\mathbf{F}'(\mathbf{x}) = \hat{\mathbf{f}}'(\mathbf{x})^{\mathsf{T}}\hat{\mathbf{f}}(\mathbf{x}) \tag{2.8}$$

$$\mathbf{F}''(\mathbf{x}) = \hat{\mathbf{f}}'(\mathbf{x})^{\mathsf{T}}\hat{\mathbf{f}}'(\mathbf{x}) \tag{2.9}$$

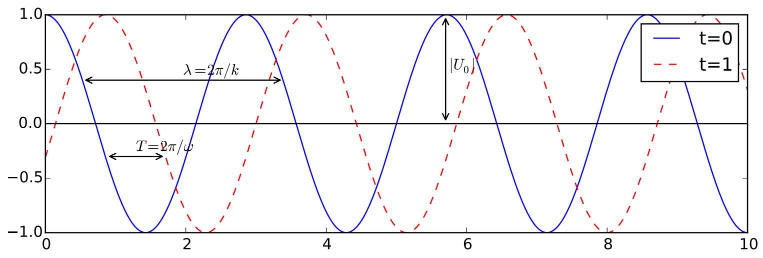


Figure 2.1: Illustration of the wave number k, the angular frequency ω and the amplitude $|U_0|$ of a plane wave $e^{i(kx-\omega t)}$.

where the only difference to Equation (2.7) is that the term involving the Hessian has dropped out. Finally, if we plug Equations (2.8) and (2.9) into Equation (2.4) we get the *Gauss-Newton* step

$$\Delta \mathbf{x} = -\left[\hat{\mathbf{f}}'(\mathbf{x})^{\mathsf{T}}\hat{\mathbf{f}}'(\mathbf{x})\right]^{-1}\hat{\mathbf{f}}'(\mathbf{x})^{\mathsf{T}}\hat{\mathbf{f}}(\mathbf{x}).$$

The approximation we made by using $\hat{\mathbf{f}}$ in place of $\mathbf{f}$ corresponds to a linearization of $\mathbf{f}(\mathbf{x}+\Delta\mathbf{x})$ about $\mathbf{x}$ and in general we can no longer expect quadratic convergence. In practice, Gauss-Newton often performs much better than gradient descent and sometimes even similarly to Newton, especially if $\mathbf{f}$ is not *too* non-linear. We apply Gauss-Newton to our non-linear deformation energy in Chapter 3.

2.2 Plane waves

Plane waves are a useful mathematical model for describing oscillations in time and space. They are ubiquitous throughout physics, they show up as solutions to certain wave-like linear differential equations (*cf.* Section 2.3) and they are crucial to the understanding of the perfectly matched layer technique that we will be investigating further in Section 2.9.

In n spatial dimensions and one time dimension, a plane wave is defined as a function

$$u(\mathbf{x}, t) = U_0 e^{i(\mathbf{k}\cdot\mathbf{x}-\omega t)} \tag{2.10}$$

where $\mathbf{k} \in \mathbb{R}^n$ is the *wave vector*, $\omega \in \mathbb{R}$ is the *angular velocity* and U_0 is the possibly complex wave *amplitude*. The direction $\mathbf{k}/||\mathbf{k}||$ of $\mathbf{k}$ points in direction normal to the surfaces of constant phase, and the magnitude $||\mathbf{k}||$ is the *wavenumber* and measures how fast the wave oscillates in space. Similarly, the *angular frequency* $|\omega|$ measures how fast u oscillates in time at a fixed point in space. The real part $\text{Re}\{U_0\}$ of the wave amplitude measures the peek of u, and the imaginary part $\text{Im}\{U_0\}$ corresponds to a phase shift. For an illustration of these quantities for plane wave in one spatial dimension, see Figure 2.1.

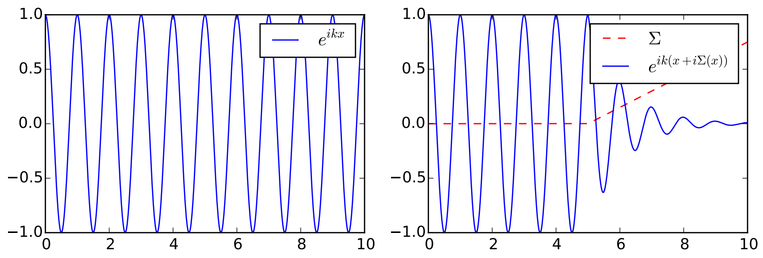


Figure 2.2: Evaluating a plane wave (for $t = 0$) along the contour $z = x + i\Sigma(x)$. Notice that the original solution (left) is unchanged as long as we evaluate the plane wave along the real line (*i.e.* when $\Sigma = 0$), but it is exponentially decaying (right) when $\Sigma > 0$.

Equation (2.10) has many nice properties, one of them being that it is an *analytic function* of $\mathbf{x}$ and t. This means that we are free to perform an *analytical continuation* of u from its original real domain into the complex plane. Although Equation (2.10) is only considered to be a plane wave when $\mathbf{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}$ and U_0 is independent of $\mathbf{x}$ and t, it is nevertheless instructive to see what happens when we evaluate a plane wave (in one spatial dimension for simplicity) along a complex contour $z = a + ib$ with $a, b \in \mathbb{R}$

$$U_0 e^{i(kz-\omega t)} = U_0 e^{i(k(a+bi)-\omega t)} = U_0 e^{i(ka-\omega t)} e^{-kb}.$$

We observe that we get a plane wave multiplied by an exponentially decaying or increasing function depending on the sign of b. It is worth emphasizing that analytic continuation does *not* change u when evaluated at points in the original (real) domain. We are merely extending the domain. For an illustration of this fact, see Figure 2.2. This fact will be important when we get to the theory of perfectly matched layers in Section 2.9.

2.3 Linear differential equations

The literature on differential equations is staggering and a myriad of different approaches exists on how to solve them. Naturally, we will not attempt to give any sort of comprehensive review here, but instead we will try to show a few useful techniques to solve a certain class of equations. These will be useful later when we try to recover velocity from vorticity in Section 2.8.1 and discuss non-reflecting boundaries in Section 2.9. The material in this section is loosely based on the contents of Palais [2000] and Wikipedia [2016].

In this section we will be using *multi-index* notation, which is straightforward to define. Let $\alpha = (\alpha_1, \cdots, \alpha_n)$ be a tuple of non-negative integers (some can be zero) called the *multi-index*. We define the *length* of α as $|\alpha| = \alpha_1 + \cdots + \alpha_n$. Also, we define exponentiation as $a^\alpha = a^{\alpha_1} \cdots a^{\alpha_n}$.

Let $\mathbf{u}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ be a vector-valued function of $\mathbf{x} = (x_1, \cdots, x_n)$, then an inhomogeneous system of N-th order linear partial differential equations can be expressed

$$P(D)\mathbf{u} = \mathbf{f} \tag{2.11}$$

where $P(X) = \sum_{|\alpha| \le N} \mathbf{A}_\alpha X^\alpha$ is a polynomial in $X = (X_1, \cdots, X_n)$ with coefficients $\mathbf{A}_\alpha$ in the space of linear operators ($m \times m$ matrices) and $D = (\frac{\partial}{\partial x_i}, \cdots, \frac{\partial}{\partial x_n})$. Since any (system of) linear differential equation(s) can be converted into a system of first-order equations by the introduction of new variables, we may equivalently consider the linear function $L(X) = \sum_{i=1}^n \mathbf{A}_i X_i$ and the first-order differential equation

$$L(D)\mathbf{u} = \mathbf{f} \tag{2.12}$$

where we have redefined $\mathbf{u}$ and $\mathbf{f}$ to accommodate the extra variables.

2.3.1 The wave equation

The definitions in the previous section may seem a bit abstract, so before we move on we will give an example. Consider the two-dimensional *wave equation*

$$\frac{\partial^2 u}{\partial t^2} - c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = \mathbf{0}.$$

It can be expressed as in Equation (2.11) by defining $X = (t, x, y)$ so that

$$\begin{aligned} P(D) &= \sum_{|\alpha| \le 2} \mathbf{A}_\alpha X^\alpha \\ &= \mathbf{A}_{(2,0,0)} D^{(2,0,0)} + \mathbf{A}_{(0,2,0)} D^{(0,2,0)} + \mathbf{A}_{(0,0,2)} D^{(0,0,2)} \\ &= \frac{\partial^2}{\partial t^2} - c^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) \end{aligned}$$

where $\mathbf{A}_{(2,0,0)} = 1$ and $\mathbf{A}_{(0,2,0)} = \mathbf{A}_{(0,0,2)} = -c^2$. Alternatively, we can express the wave equation as in Equation (2.12) by letting $\mathbf{u} = [u, \mathbf{v}]^\top$ where u can be interpreted as pressure and $\mathbf{v}$ as velocity and

$$L(D) = \begin{bmatrix} \frac{\partial}{\partial t} & -c^2\nabla\cdot \\ -\nabla & \frac{\partial}{\partial t} \end{bmatrix}$$

or even more explicitly with $\mathbf{u} = [u, v, w]^\top$

$$L(D) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{\partial}{\partial t} + \begin{bmatrix} 0 & -c^2 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{\partial}{\partial x} + \begin{bmatrix} 0 & 0 & -c^2 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \frac{\partial}{\partial y}.$$

2.3.2 Green's functions

One way of solving an equation like Equation (2.11) is to find a right inverse to $P(D)$ such that

$$P(D)\,G(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x}'-\mathbf{x}).$$

where δ is the n-dimensional Dirac delta function. Such a right-inverse $G(\mathbf{x},\mathbf{x}')$ is called a *Green's function* to the linear differential operator $P(D)$. Given G we can obtain a new expression for the right-hand side of Equation (2.11)

$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \int \delta(\mathbf{x}'-\mathbf{x})\mathbf{f}(\mathbf{x}')\,\mathrm{d}\mathbf{x}' \\ &= \int P(D)\,G(\mathbf{x},\mathbf{x}')\mathbf{f}(\mathbf{x}')\,\mathrm{d}\mathbf{x}' \\ &= P(D)\int G(\mathbf{x},\mathbf{x}')\mathbf{f}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'\end{aligned} \tag{2.13}$$

where the last equality follows from the fact that $P(D)$ does not depend on the integration variable $\mathbf{x}'$ and can thus be pulled outside the integral. Substituting Equation (2.13) into Equation (2.11) we get

$$P(D)\,\mathbf{u}(\mathbf{x}) = P(D)\int G(\mathbf{x},\mathbf{x}')\mathbf{f}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'$$

which suggests that

$$\mathbf{u}(\mathbf{x}) = \int G(\mathbf{x},\mathbf{x}')\mathbf{f}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'$$

Finding a Green's function for a given linear differential operator is in general a non-trivial task and we will not explain how to do it here. Instead, we will give (without proof) the Green's function to one of the most frequently occuring operators in computer graphics, the *three*-dimensional scalar Laplacian $\nabla^2 := \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$. It is given by

$$G(\mathbf{x},\mathbf{x}') = -\frac{1}{4\pi}\frac{1}{|\mathbf{x}-\mathbf{x}'|}. \tag{2.14}$$

The Laplacian appears in a myriad of important equations, such as the heat equation, the wave equation and, perhaps the most frequently occurring equation in computer graphics, the Poisson equation

$$\nabla^2 u(\mathbf{x}) = f(\mathbf{x}) \tag{2.15}$$

Using the Green's function in Equation (2.14) one possible way of solving Equation (2.15) is thus given by the integral

$$u(\mathbf{x}) = -\frac{1}{4\pi}\int_{\mathbb{R}^3} \frac{f(\mathbf{x}')}{|\mathbf{x}-\mathbf{x}'|}\,\mathrm{d}\mathbf{x}' \tag{2.16}$$

which is the so-called *free space* (vanishing boundaries at infinity) solution to the *three*-dimensional (because we used the three-dimensional Green's function) Poisson equation in Equation (2.15).

2.3.3 Fourier methods

In this section we will look at how the Fourier transform can be used to solve linear differential equations. Recall that the *spatial* Fourier transform of the function $\mathbf{f}(\mathbf{x},t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^m$ that takes the spatial variable $\mathbf{x} = (x_1, \cdots, x_n)$ to the wave vector $\mathbf{k} = (k_1, \cdots, k_n)$ is defined

$$\widetilde{\mathbf{f}}(\mathbf{k},t) = (2\pi)^{-n/2} \int\limits_{\mathbb{R}^n} \mathbf{f}(\mathbf{x},t) e^{-i\mathbf{k}\cdot\mathbf{x}} \, d\mathbf{x} \tag{2.17}$$

Similarly, the inverse transform is defined

$$\mathbf{f}(\mathbf{x},t) = (2\pi)^{-n/2} \int\limits_{\mathbb{R}^n} \widetilde{\mathbf{f}}(\mathbf{k},t) e^{i\mathbf{k}\cdot\mathbf{x}} \, d\mathbf{k} \tag{2.18}$$

Consider the homogeneous linear evolution equation

$$\frac{\partial \mathbf{u}}{\partial t} + P(D)\,\mathbf{u} = \mathbf{0} \tag{2.19}$$

where $\mathbf{u}(\mathbf{x},t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^m$ is a time-dependent vector-valued function. This is still of the form in Equation (2.11), only we have made the time-dependence explicit by separating out the time variable t. If we assume that that $P(D)$ has constant coefficients, we can apply Equation (2.17) to both sides of Equation (2.19) and obtain

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial t} + P(ik)\,\widetilde{\mathbf{u}} = \mathbf{0} \tag{2.20}$$

In the frequency domain Equation (2.20) is now an ordinary differential equation, which we can solve analytically

$$\widetilde{\mathbf{u}}(\mathbf{k},t) = \widetilde{\mathbf{u}_0}(\mathbf{k}) e^{-t\,P(ik)} \tag{2.21}$$

with $\widetilde{\mathbf{u}_0}(\mathbf{k})$ being some initial condition. To obtain a solution $\mathbf{u}(\mathbf{x},t)$ in the original domain, all that remains is to plug Equation (2.21) into the inverse Fourier transform in Equation (2.18)

$$\begin{aligned}
\mathbf{u}(\mathbf{x},t) &= (2\pi)^{-n/2} \int\limits_{\mathbb{R}^n} \widetilde{\mathbf{u}_0}(\mathbf{k}) e^{-t\,P(ik)} e^{i\mathbf{k}\cdot\mathbf{x}} \, d\mathbf{k} \\
&= (2\pi)^{-n/2} \int\limits_{\mathbb{R}^n} \widetilde{\mathbf{u}_0}(\mathbf{k}) e^{i\left((\mathbf{k}\cdot\mathbf{x})\mathbf{I}_m - \frac{P(ik)}{i} t\right)} \, d\mathbf{k} \\
&= (2\pi)^{-n/2} \int\limits_{\mathbb{R}^n} \widetilde{\mathbf{u}_0}(\mathbf{k}) e^{i((\mathbf{k}\cdot\mathbf{x})\mathbf{I}_m - \omega t)} \, d\mathbf{k}
\end{aligned} \tag{2.22}$$

where $\mathbf{I}_m$ is the m-dimensional identity matrix and we have defined the *dispersion relation* $\omega(k) \overset{\text{def}}{=} \frac{P(ik)}{i}$.

If $\mathbf{u}$ is m-dimensional then ω will be an $m \times m$ (complex) matrix. If we eigendecompose this matrix we get $\omega = \mathbf{Q}\boldsymbol{\Omega}\mathbf{Q}^{-1}$ where $\mathbf{Q}$ is the matrix of eigenvectors and $\boldsymbol{\Omega} = \operatorname{diag}(\omega_1, \cdots, \omega_m)$ is the diagonal matrix of eigenvalues. Using a few facts about matrix exponentials we further see that

$$e^{-i\omega t} = e^{-i\mathbf{Q}\boldsymbol{\Omega}\mathbf{Q}^{-1}t} = \mathbf{Q}e^{-i\boldsymbol{\Omega}t}\mathbf{Q}^{-1} = \mathbf{Q}\operatorname{diag}\left(e^{-i\omega_1 t}, \cdots, e^{-i\omega_m t}\right)\mathbf{Q}^{-1}$$

If we assume that $P(D)$ is a skew-adjoint differential operator then $P(ik)$ is anti-Hermitian and ω Hermitian, which means that it eigendecomposes as $\omega = \mathbf{U}\Omega\mathbf{U}^\star$ with unitary $\mathbf{U}$ and real Ω. This suggests that Equation (2.22) is a superposition of plane waves.

2.3.4 Numerical methods

Given a linear differential equation like Equation (2.11) it is also always possible to approximate it numerically. This is done by replacing all derivatives with algebraic expressions and solving the resulting linear system of algebraic equations. Standard methods include finite differences (FD), the Finite Element Method (FEM) and boundary elements. Fluid simulation practitioners have historically favored finite difference methods, so we give a quick summary in this section for completeness.

Let $u(x, y)$ be a two-dimensional scalar function and consider the first-order Taylor expansion

$$u(x + \Delta x, y) = u(x, y) + \frac{\partial u(x, y)}{\partial x}\Delta x + O\left(\Delta x^2\right).$$

This equation can be rearranged to give an expression for the partial derivative with respect to x

$$\frac{\partial u(x, y)}{\partial x} = \frac{u(x + \Delta x, y) - u(x, y)}{\Delta x} + O\left(\Delta x^2\right).$$

If we truncate the second-order terms in this equation, we arrive at a first-order approximation to $\partial u/\partial x$. Such an approximation is called a *forward difference* or *forward Euler* in the case where x is the time variable. It is possible to make a second-order approximation

$$\frac{\partial u(x, y)}{\partial x} = \frac{u(x + \Delta x, y) - u(x - \Delta x, y)}{2\Delta x} + O\left(\Delta x^3\right)$$

which is called a *central difference*. By iterating these kinds of finite difference approximations it is possible to obtain expressions for higher derivatives. Finally, lets see an example of how to build a finite difference approximation to the two-dimensional Poisson equation from Equation (2.15). Let us assume that u and f

have been discretized unto a two-dimensional grid with equal spacing Δx in each dimension. Let $u_{i,j} = u(i\Delta x, j\Delta x)$, then the equation for cell (i,j) is

$$\frac{-4u_{i,j} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{\Delta x^2} = f_{i,j}$$

which can be assembled into a system of linear equations

$$\frac{1}{\Delta x^2}\begin{pmatrix} & & & \vdots & & & \\ & & & & & & \\ & & & & & & \\ \cdots & 1 & 1 & -4 & 1 & 1 & \cdots \\ & & & & & & \\ & & & & & & \\ & & & \vdots & & & \end{pmatrix}\begin{pmatrix} \vdots \\ u_{i+1,j} \\ u_{i-1,j} \\ u_{i,j} \\ u_{i,j+1} \\ u_{i,j-1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \\ \\ f_{i,j} \\ \\ \\ \vdots \end{pmatrix} \tag{2.23}$$

or more compactly as

$$\mathbf{A}\mathbf{u} = \mathbf{f}$$

This linear system can be inverted to find a solution. In the case of a discretized Poisson equation in Equation (2.23), $\mathbf{A}$ is a sparse symmetric negative definite matrix and can be inverted very efficiently using either a direct solver based on Cholesky decomposition or, for larger problems, using an iterative solver such as conjugate gradients [Shewchuk 1994]. See Botsch et al. [2005] for a survey of linear system solvers for geometry processing.

2.4 Local description of velocity

Given a time-dependent velocity field $\mathbf{u}(\mathbf{x}, t)$ we may look at how it behaves locally by considering a first-order Taylor approximation around a point $\mathbf{x}$

$$\mathbf{u}(\mathbf{x} + \Delta\mathbf{x}, t) = \mathbf{u}(\mathbf{x}, t) + \nabla\mathbf{u}(\mathbf{x}, t)\Delta\mathbf{x} + O\big(\|\Delta\mathbf{x}\|^2\big)$$

We may further split $\nabla\mathbf{u}$ into its symmetric and anti-symmetric parts

$$\begin{aligned} \mathbf{u}(\mathbf{x} + \Delta\mathbf{x}, t) &\approx \mathbf{u}(\mathbf{x}, t) + \nabla\mathbf{u}(\mathbf{x}, t)\Delta\mathbf{x} \\ &= \mathbf{u}(\mathbf{x}, t) + \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^\mathsf{T})\Delta\mathbf{x} + \frac{1}{2}(\nabla\mathbf{u} - \nabla\mathbf{u}^\mathsf{T})\Delta\mathbf{x} \end{aligned}$$

Letting $\varepsilon = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^\mathsf{T})$ be the symmetric part and $\omega = \frac{1}{2}(\nabla\mathbf{u} - \nabla\mathbf{u}^\mathsf{T})$ the anti-symmetric part, we may simplify further

$$\mathbf{u}(\mathbf{x} + \Delta\mathbf{x}, t) \approx \mathbf{u}(\mathbf{x}, t) + (\varepsilon + \omega)\Delta\mathbf{x}.$$

The infinitesimal strain tensor ε measures the local deformation of $\mathbf{u}$ while the infinitesimal rotation tensor ω measures the local rotation. In slightly non-standard notation, let $\xi = \nabla \times \mathbf{u}$ be the *vorticity* (not to be confused with the angular velocity $\boldsymbol{\omega}$). It is easy to verify that

$$2\omega\Delta\mathbf{x} = 2\boldsymbol{\omega} \times \Delta\mathbf{x} = \xi \times \Delta\mathbf{x}$$

so vorticity equals twice the local angular velocity. We go into more detail on vorticity in Section 2.8.

2.5 Reynold's transport theorem

In the next section we will derive the equations of motion for a continuum. In doing so we will be making heavy use of *Reynold's transport theorem*. Let $\Omega(t)$ be an arbitrary non-zero region of space that is being advected by a velocity field $\mathbf{u}(\mathbf{x},t)$. The time derivative of a quantity $\phi(\mathbf{x},t)$ integrated over $\Omega(t)$ is then given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega(t)} \phi(\mathbf{x},t)\,\mathrm{d}V = \int\limits_{\Omega(t)} \left(\frac{\partial\phi}{\partial t} + \nabla\cdot(\phi\mathbf{u})\right)\mathrm{d}V \tag{2.24}$$

where $\mathrm{d}V$ denotes the volume element.

2.6 Continuum mechanics

Given a non-zero and arbitrary region of space $\Omega(t)$ we get its mass by integrating density $\rho(\mathbf{x},t)$ over the volume $M = \int_{\Omega(t)} \rho\,\mathrm{d}V$. A basic law of physics is the law of conservation of mass. Using Reynold's transport Theorem we get

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}M = \frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega(t)} \rho\,\mathrm{d}V = \int\limits_{\Omega(t)} \left(\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u})\right)\mathrm{d}V$$

The only function that integrates to zero for arbitrary $\Omega(t)$ is zero itself, so we may remove the integral

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0 \tag{2.25}$$

which is called the *continuity equation*.

From Newton's second law we know that the change in momentum of $\Omega(t)$ is equal to the net forces acting on $\Omega(t)$. These forces include contact forces acting on the surface of $\Omega(t)$ and body forces such as gravity that act at every point in $\Omega(t)$. The forces acting on the surface of $\Omega(t)$ are called tractions $\mathbf{T}(\mathbf{x})$ and are given by $\mathbf{T}(\mathbf{x}) = \sigma(\mathbf{x})\cdot\hat{\mathbf{n}}$ where σ is a second-order tensor called *Cauchy's stress tensor* and $\hat{\mathbf{n}}$ is the surface normal to the surface $\partial\Omega(t)$ of $\Omega(t)$. Tractions have units of force per unit area. Body forces are given by the force density $\mathbf{f}(\mathbf{x}) = \rho(\mathbf{x})\mathbf{g}(\mathbf{x})$ where $\mathbf{g}$ is acceleration per unit volume. Force density has units of force per unit volume.

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega(t)} \rho\mathbf{u}\,\mathrm{d}V = \int\limits_{\partial\Omega(t)} \sigma\cdot\hat{\mathbf{n}}\,\mathrm{d}S + \int\limits_{\Omega(t)} \mathbf{f}\,\mathrm{d}V \tag{2.26}$$

where $\partial\Omega(t)$ is the boundary of $\Omega(t)$ and $\hat{\mathbf{n}}$ is the outward pointing normal to $\partial\Omega(t)$. Then we apply Reynold's Transport Theorem to the left-hand side

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega(t)} \rho\mathbf{u}\,\mathrm{d}V &= \int\limits_{\Omega(t)} \left(\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u})\right)\mathrm{d}V \\
&= \int\limits_{\Omega(t)} \left(\frac{\partial\mathbf{u}}{\partial t}\rho + \frac{\partial\rho}{\partial t}\mathbf{u}\right)\mathrm{d}V + \int\limits_{\Omega(t)} \Big(\nabla\cdot(\rho\mathbf{u})\mathbf{u} + (\mathbf{u}\cdot\nabla\mathbf{u})\rho\Big)\mathrm{d}V \\
&= \int\limits_{\Omega(t)} \cancelto{0}{\left(\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u})\right)}\mathbf{u}\,\mathrm{d}V + \int\limits_{\Omega(t)} \cancelto{\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}}{\left(\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right)}\rho\,\mathrm{d}V \\
&= \int\limits_{\Omega(t)} \rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}\,\mathrm{d}V
\end{aligned}$$

where the first cancellation comes from Equation (2.25) and $\mathrm{D}/\mathrm{D}t$ is the material derivative. Inserting this result back into Equation (2.26) and converting the surface integral to a volume integral through the divergence theorem we get

$$\int\limits_{\Omega(t)} \rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}\,\mathrm{d}V = \int\limits_{\Omega(t)} (\nabla\cdot\sigma + \mathbf{f})\,\mathrm{d}V$$

Finally, we again note that since $\Omega(t)$ was arbitrary the integrands much be equal and we arrive at *Cauchy's momentum equation*

$$\rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \nabla\cdot\sigma + \mathbf{f}. \tag{2.27}$$

2.7 Fluid dynamics

The equations of motion for a viscous fluid are called the *Navier-Stokes* equations. We will derive these equations from Equations (2.25) and (2.27) in the following, but before we do so we will make a further simplifying assumption. The fluids of interest in computer graphics, such as smoke and liquids at speed far below the speed of sound do not change their volumes easily, so it convenient to make an assumption of *incompressible flow*. Incompressible flow is any flow that keeps volumes constant, *i.e.* $\mathrm{d}V/\mathrm{d}t = 0$. It follows from Reynold's Transport theorem that

$$0 = \frac{\mathrm{d}V}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega(t)} 1\,\mathrm{d}V = \int\limits_{\Omega(t)} \left(\cancelto{0}{\frac{\partial 1}{\partial t}} + \nabla\cdot\mathbf{u}\right)\mathrm{d}V$$

which, following similar reasoning as used in the previous section, implies that

$$\nabla\cdot\mathbf{u} = 0 \tag{2.28}$$

which is called the *incompressibility constraint* and is the first equation in the *incompressible* Navier-Stokes equations. Equation (2.28) is equivalent to assuming that density is advected passively with the flow, *i.e.* $\mathrm{D}\rho/\mathrm{D}t = 0$, which is easily seen from Equation (2.25).

To derive the other equation of incompressible Navier-Stokes, we need to specify a material model (also called a constitutive model). First, we will split the stress tensor into two parts

$$\sigma = -p\delta + \tau \tag{2.29}$$

where δ is the identity tensor. We will use the pressure p to enforce the incompressibility constraint and the viscous stress tensor τ to model viscosity. We will use a linear stress-strain relation for τ

$$\tau = 2\mu\varepsilon + \lambda\,\mathrm{tr}(\varepsilon)\delta.$$

The dynamic viscosity coefficient μ controls the resistance to shearing and the bulk viscosity coefficient λ controls viscosity due to compression and dilation. The infinitesimal strain tensor $\varepsilon = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^{\mathsf{T}})$ is as we saw in Section 2.4. Intuitively, the trace $\mathrm{tr}(\varepsilon)$ measures the local volume change, which is zero for incompressible flow, *i.e.* $\mathrm{tr}(\varepsilon) = \nabla\cdot\mathbf{u} = 0$, so it is customary to set $\lambda = 0$. If we further assume that μ is constant in space and plug σ into Equation (2.27) we obtain

$$\rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} + \nabla p - \mu\nabla^2\mathbf{u} = \mathbf{f}$$

which is the *momentum equation* of Navier-Stokes. If we further expand the material derivative $\mathrm{D}/\mathrm{D}t$, divide through by ρ (recalling that $\mathbf{f} = \rho\mathbf{g}$) and define the kinematic viscosity $\nu = \mu/\rho$ we get

$$\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{1}{\rho}\nabla p - \nu\nabla^2\mathbf{u} = \mathbf{g}. \tag{2.30}$$

which is the form most commonly seen in computer graphics literature.

Equations (2.28) and (2.30) together form the incompressible Navier-Stokes equations. However, we still need to relate the pressure in the momentum equation to the incompressibility constraint, which we will do in the next section.

2.7.1 Pressure

The pressure appearing in the momentum equation (Equation (2.30)) is somewhat strange in the case of incompressible flow. The pressure itself is *not* a physical quantity, but the gradient of pressure ∇p is a force. In Chapter 4 we will be using the pressure gradient to define an energy that measures deviations of a liquid surface from the set of physically valid states, so it makes sense to spend a little time here to try and understand pressure.

First, lets derive an expression for pressure. We will assume the incompressibility constraint $\nabla \cdot \mathbf{u} = 0$ and see what we get. By taking the divergence of the momentum equation we get

$$\nabla \cdot \frac{1}{\rho} \nabla p = \nabla \cdot \big(-\, \partial \mathbf{u} / \partial t \, - (\mathbf{u} \cdot \nabla)\, \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{g}. \big) \tag{2.31}$$

If we now use the incompressibility assumption so (assuming derivatives commute) $\nabla \cdot \left(\frac{\partial \mathbf{u}}{\partial t} \right) = \frac{\partial}{\partial t}(\nabla \cdot \mathbf{u}) = 0$ we obtain

$$\nabla \cdot \frac{1}{\rho} \nabla p = \nabla \cdot \big(-(\mathbf{u} \cdot \nabla)\, \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{g} \big) \tag{2.32}$$

which is a (variable coefficient) Poisson equation.

We will now show that the pressure in Equation (2.32) *enforces* the incompressibility constraint. Without assuming $\nabla \cdot \mathbf{u} = 0$ we substitute the expression for pressure in Equation (2.32) into Equation (2.31) to obtain (after commuting derivatives)

$$\frac{\partial}{\partial t}(\nabla \cdot \mathbf{u}) = 0$$

If we integrate this equation in time we get

$$\nabla \cdot \mathbf{u} = g(\mathbf{x})$$

which is independent of time. If we now assume that the initial divergence is zero this implies that $g(\mathbf{x}) = 0$ for all time. This shows that the pressure in Equation (2.32) does indeed imply that incompressibility is enforced. In Appendix A we show how Equation (2.32) can be solved numerically using methods from Section 2.3 and be used to enforce the incompressibility constraint.

What we have shown is section is that *any* divergence-free velocity field induces a pressure through the incompressible Navier-Stokes equations and that this pressure ensures that the accelerations in Navier-Stokes are divergence-free *guaranteeing* that the velocity field remains divergence-free for *all* time. See Gresho and Sani [1987] for a more detailed discussion.

2.7.2 Helmholtz-Hodge Decomposition Theorem

Another way of looking at pressure is as a projection onto the set of divergence-free vector fields through the Helmholtz-Hodge decomposition.

If we let Ω be a subset of $\mathbb{R}^3$ with smooth boundary $\partial\Omega$ then the Helmholtz-Hodge decomposition theorem states that *any* smooth vector field $\mathbf{u} : \Omega \to \mathbb{R}^3$ can be decomposed *uniquely* as

$$\mathbf{u} = \nabla \phi + \nabla \times \boldsymbol{\Psi} + \boldsymbol{\gamma} \tag{2.33}$$

where $\phi : \Omega \to \mathbb{R}$ is a scalar field, $\mathbf{\Psi} : \Omega \to \mathbb{R}^3$ is a *divergence-free*[1] vector field (meaning $\nabla \cdot \mathbf{\Psi} = 0$) and $\boldsymbol{\gamma} : \Omega \to \mathbb{R}^3$ is a *harmonic* vector-valued field (meaning $\nabla \cdot \boldsymbol{\gamma} = 0$ and $\nabla \times \boldsymbol{\gamma} = 0$) [Cantarella et al. 2002].

To extract the divergence-free part of $\mathbf{u}$, let us take the divergence on both sides of Equation (2.33)

$$\nabla \cdot \mathbf{u} = \nabla \cdot \nabla \phi = \nabla^2 \phi$$

which gives us a Poisson equation for ϕ.

We can now define a projection operator $\mathbb{P}(\mathbf{u}) = \mathbf{u} - \nabla\phi$, which extracts the divergence-free part. Clearly, $\nabla \cdot \mathbb{P}(\mathbf{u}) = 0$ and $\mathbb{P}(\mathbb{P}(\mathbf{u})) = \mathbb{P}(\mathbf{u})$ so $\mathbb{P}$ is a proper projection. Notice that these steps correspond exactly to the steps we took in Section 2.7.1 with pressure taking the place of ϕ and the momentum equation taking the place of $\mathbf{u}$.

Alternatively, we can take the curl of Equation (2.33) and get

$$\begin{aligned} \nabla \times \mathbf{u} &= \nabla \times \nabla \times \mathbf{\Psi} \\ &= \nabla\cancelto{0}{(\nabla \cdot \mathbf{\Psi})} - \nabla^2 \mathbf{\Psi} \\ &= -\nabla^2 \mathbf{\Psi} \end{aligned} \tag{2.34}$$

where the second equality follows from a vector calculus identity and the third equality from the fact that $\nabla \cdot \mathbf{\Psi} = 0$ in the decomposition. Notice that Equation (2.34) is a vector-Poisson equation for $\mathbf{\Psi}$.

For Ω with simple topology, where we may assume that $\boldsymbol{\gamma} = 0$, we can again define a projection operator $\mathbb{P}(\mathbf{u}) = \nabla \times \mathbf{\Psi}$ to recover the divergence-free part of $\mathbf{u}$. Even when Ω has non-trivial topology, $\boldsymbol{\gamma}$ is often small for the kinds of vector fields that arise from fluid flows [Tong et al. 2003]. This means that in many fluid simulation papers in computer graphics (*e.g.* Ando et al. [2015]) the authors apply this method even when Ω has non-trivial topology.

2.7.3 Boundary conditions

So far we have only talked about what happens in the interior of the fluid and have carefully avoided any mention of boundaries. In this section we will answer which kinds of boundaries exists between a fluid and its surrounding environment, and what happens at these boundaries. This gives us a chance to talk about two important concept, namely surface tension and free surfaces.

In the following we will denote by $\partial\Omega$ the boundary of the fluid. There are two different kinds of boundaries to consider. First, there are *solid-fluid* boundaries $\partial\Omega_S$ where the fluid is in contact with a rigid solid. Second, there are *fluid-fluid*

[1]Divergence-free velocity fields are also sometimes called *solenoidal* vector fields.

boundaries $\partial\Omega_\mathrm{F}$ where two fluids are in contact. The two kinds of boundaries together form the whole boundary of the fluid $\partial\Omega = \partial\Omega_\mathrm{S} \cup \partial\Omega_\mathrm{F}$.

For the purposes of this thesis we will be assuming that when multiple fluids interact, that the density of one of the fluids is significantly higher than the density of the other fluids. For instance, this is the case of water and air that have a density ratio of about one thousand. Under this assumption we may neglect the influence of the lighter fluids on the motion of the heavier fluid. This approximation is called the *free surface approximation*[2] and is prevalent in computer graphics, since it allows us to model only a single fluid with significant computational savings as a result.

Even when we assume that fluid-fluid interfaces are free surfaces, there is another effect that becomes important at such interfaces at small length scales and that is *surface tension*. Surface tension is a cohesive force resulting from the fact that fluid particles prefer to stick together with other fluid particles from the same fluid. Surface tensions causes a fluid to tend towards reducing its surface area, which causes extra tension at the surface.

We will not dwell too much on topic, but as was the case when we derived the equations of motion for the interior of a continuum in Section 2.6, what happens at the boundary is also given by a force balance. In this case the net force on the area element of the surface should be balanced which eventually leads to the following boundary condition (see Subramanian [2015] and Bridson [2008] for more details) for the stress

$$(\sigma_1 - \sigma_2)\hat{\mathbf{n}} = 2H\gamma\hat{\mathbf{n}} \quad \text{on} \quad \partial\Omega_\mathrm{F}$$

where σ_1 and σ_2 are the stresses for the two fluids that form the interface, $\hat{\mathbf{n}}$ is the surface normal to the interface and H is the mean curvature of the interface (we will see how to compute this in Section 2.10). Surface tension is denoted by γ and has units of force per unit length. For the interface between water and air at room temperature it is approximately $\gamma = 0.073$ newton per meter. Recalling from Equation (2.29) that the stress was defined as $\sigma = -p\delta + \tau$, this means that for an *inviscid* fluid with no viscosity (*i.e.* $\tau = 0$) we have

$$(p_1 - p_2) = 2H\gamma \quad \text{on} \quad \partial\Omega_\mathrm{F}$$

since pressure only works in the normal direction (it was defined as a diagonal stress tensor). Under the free surface approximation we may simplify this further. We could for instance set p_2 to be the ambient air pressure. In fact, since pressure is only ever evaluated as a gradient, constant offsets do not matter and we may equivalently set $p_2 = 0$ so (defining $p := p_1$)

$$p = 2H\gamma \quad \text{on} \quad \partial\Omega_\mathrm{F}$$

[2]Technically, a free surface is a surface subject to constant normal stress and zero shear stress. This prevents effects like the wind creating waves in the ocean, which is a shear effect.

Finally, in case we are mainly interested in large scales (*e.g.* if we are modelling the ocean) we may neglect surface tension completely, which gives us

$$p = 0 \quad \text{on} \quad \partial\Omega_{\text{F}}$$

At solid-fluid boundaries we certainly do not want fluid to flow into or out of the solid. This leads to the following boundary condition

$$\mathbf{u} \cdot \hat{\mathbf{n}} = \mathbf{u}_{\text{solid}} \cdot \hat{\mathbf{n}} \quad \text{on} \quad \partial\Omega_{\text{S}}.$$

where where $\mathbf{u}_{\text{solid}}$ is the velocity of the solid and $\hat{\mathbf{n}}$ is the normal vector to solid obstacle. This boundary condition is called *free-slip*, since the fluid is unrestricted in its movement tangentially to the solid. This is the correct boundary condition for an inviscid fluid. For a viscous fluid it turns out that the correct boundary condition is

$$\mathbf{u} = \mathbf{u}_{\text{solid}} \quad \text{on} \quad \partial\Omega_{\text{S}}.$$

This boundary condition is called *no-slip*, since it forces the fluid to have the same velocity as the solid at the boundary. Nevertheless, it is common for practioners to use the free-slip boundary condition even for viscous fluids, since the no-slip boundary condition, although accurate in the limit, is too inaccurate for the kinds of coarse grids used in computer graphics.

2.7.4 Conservation form

In Chapter 5 we will be using an alternative but equivalent version of Navier-Stokes for convenience. This form of Navier-Stokes is called *conservation form* and emphasizes the fact that quantities like momentum and mass are conserved.

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{f} \tag{2.35}$$

$$\mathbf{q} = \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix}, \quad \mathbf{F}(\mathbf{q}) = \begin{pmatrix} \mathbf{u} \\ \mathbf{u} \otimes \mathbf{u} - \frac{1}{\rho}\sigma \end{pmatrix} \tag{2.36}$$

Here, $\mathbf{u}$ denotes the usual three-dimensional velocity and σ is Cauchy's strain tensor as defined in Equation (2.29). Notice that we include both the momentum equation and the incompressibility constraint into *one* equation.

To see why Equation (2.35) is called conservation form, let $\mathbf{f} = \mathbf{0}$, integrate both sides over a fixed volume Ω and apply the divergence theorem to obtain

$$\frac{\partial}{\partial t} \int\limits_{\Omega} \mathbf{q} \, \mathrm{d}V + \int\limits_{\partial\Omega} \mathbf{F}(\mathbf{q}) \cdot \hat{\mathbf{n}} \, \mathrm{d}A = \mathbf{0}$$

where $\hat{\mathbf{n}}$ is the outwards pointing normal vector to $\partial\Omega$. This says that the change of $\mathbf{q}$ in time inside Ω is purely due to the flux $\mathbf{F}(\mathbf{q})$ through $\partial\Omega$. Since Ω is arbitrary this implies that $\mathbf{q}$ is (locally) conserved. Note, this is only true in the interior of the fluid and separate zero-flux boundary conditions have to be applied to ensure that the equations form a closed system so that the quantities are conserved globally.

2.8 Vorticity

In Chapter 4 we will be deriving a surface-based energy whose minimum is attained when the water surface is physically valid. When we interpret this energy as a physical potential, we obtain equations of motion that are strikingly similar to equations of motion for *vortex sheet*, which are two-dimensional surfaces of concentrated vorticity. In Section 2.8.3 we will have a look at the vortex sheet equations, but before that let us try to understand vorticity a bit better.

In Section 2.4 we saw the definition of vorticity as the curl of velocity

$$\boldsymbol{\xi} = \nabla \times \mathbf{u}$$

We also saw that it is proportional to the (local) angular velocity $\boldsymbol{\omega}$ of the velocity. Notice that vorticity (and angular velocity) is a *vector* whose direction defines the axis of local rotation and whose magnitude specifies the speed of the rotation.

Another way of looking at this, is through the concept of *circulation*. If C is a closed curve sitting in a velocity field $\mathbf{u}$ in $\mathbb{R}^3$ then the circulation Γ around C is defined to be

$$\Gamma := \int_C \mathbf{u} \cdot \mathrm{d}\mathbf{l}$$

If $C = \partial S$ is the boundary of a surface S, then it follows from Stokes' theorem that

$$\Gamma = \int_C \mathbf{u} \cdot \mathrm{d}\mathbf{l} = \int_S \nabla \times \mathbf{u} \cdot \mathrm{d}\mathbf{S} = \int_S \boldsymbol{\xi} \cdot \mathrm{d}\mathbf{S}$$

which suggests that vorticity is circulation per unit area around an infinitesimal loop. Conversely, the flux of vorticity through S is the circulation.

From Section 2.7 we know that divergence-free velocity fields correspond to incompressible fluid flow. Therefore, one might equivalently choose to work with vorticity instead of velocity, which practitioners actually do as we will see in Sections 2.8.2 and 2.8.3. The question is, why would you? One of the main reasons is that while velocity is non-zero almost everywhere (at least for any interesting motion) vorticity can be far more concentrated. This makes sense once you consider that vorticity is a spatial derivative of velocity, which means constant or slowly varying regions of velocity correspond to (almost) zero vorticity. As we shall see in Section 2.8.1, it is also instructive to think of electromagnetism where the current through a wire (vorticity) can generate a whole magnetic (velocity) field. This has led to models for (zero-dimensional) vortex particles [Selle et al. 2005], (one-dimensional) vortex filaments [Angelidis and Neyret 2005] and (two-dimensional) vortex sheets [Kim et al. 2009; Pfaff et al. 2012] as we will see in Section 2.8.3.

2.8.1 Velocity from vorticity

In Chapter 4 we will be deriving a surface-based energy to reduce the deviation of a liquid surface from a physically valid state. We interpret the negative gradient (the direction of steepest descent) of this energy as the (local) angular velocity $\boldsymbol{\omega}$. Eventually, we have to turn these local rotations into motion of the surface. Luckily, we saw in Section 2.4 that $\boldsymbol{\omega}$ is proportional to the vorticity $\xi = \nabla \times \mathbf{u}$ of some velocity field $\mathbf{u}$. This allows us to leverage existing techniques for turning vorticity into velocity in which we may then move the surface.

One way of relating vorticity to velocity is through Equation (2.34) which gives us a vector-Poisson equation for the so-called *stream function* $\boldsymbol{\Psi}$

$$\nabla^2 \boldsymbol{\Psi} = -\xi. \tag{2.37}$$

This equation can be solved subject to $\nabla \cdot \boldsymbol{\Psi} = 0$ using methods from Section 2.3.4. The velocity can then be recovered by noticing that

$$-\xi = -\nabla \times \mathbf{u} = \nabla^2 \boldsymbol{\Psi} = -\nabla \times \nabla \times \boldsymbol{\Psi} + \nabla \cancelto{0}{(\nabla \cdot \boldsymbol{\Psi})}$$

which suggests that $\mathbf{u} = \nabla \times \boldsymbol{\Psi}$.

Since Equation (2.37) is a vector-Poisson equation, it can also be solved using Green's functions (*cf.* Section 2.3.2). Concretely, we can apply the Green's function for the three-dimensional Laplacian for each of the three dimensions independently. According to Equation (2.16) we have

$$\begin{aligned}
\mathbf{u} = \nabla \times \boldsymbol{\Psi} &= \nabla \times \left(-\frac{1}{4\pi} \int\limits_{\mathbb{R}^3} \frac{-\xi(\mathbf{x}')}{\|\mathbf{x}-\mathbf{x}'\|} \, d\mathbf{x}' \right) \\
&= \frac{1}{4\pi} \int\limits_{\mathbb{R}^3} \frac{\|\mathbf{x}-\mathbf{x}'\| \nabla \times \xi(\mathbf{x}') - \nabla \|\mathbf{x}-\mathbf{x}'\| \times \xi(\mathbf{x}')}{\|\mathbf{x}-\mathbf{x}'\|^2} \, d\mathbf{x}' \\
&= -\frac{1}{4\pi} \int\limits_{\mathbb{R}^3} \frac{(\mathbf{x}-\mathbf{x}')}{\|\mathbf{x}-\mathbf{x}'\|^3} \times \xi(\mathbf{x}') \, d\mathbf{x}'
\end{aligned} \tag{2.38}$$

where the curl ($\nabla\times$) is taken with respect to $\mathbf{x}$ so that $\nabla \times \xi(\mathbf{x}') = \mathbf{0}$. In the derivation we have additionally used the quotient rule and the identity $\nabla \|\mathbf{x}-\mathbf{x}'\| = (\mathbf{x}-\mathbf{x}')/\|\mathbf{x}-\mathbf{x}'\|$. Equation (2.38) is called the *Biot-Savart law*.

Since Equation (2.38) is singular (shoots to infinity) when evaluated where ξ is non-zero, it is common (this is also what we do) to instead use the *regularized* version [Pfaff et al. 2012]

$$\mathbf{u} = -\frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{(\mathbf{x}-\mathbf{x}')}{\left(\|\mathbf{x}-\mathbf{x}'\|^2+\varepsilon^2\right)^{\frac{3}{2}}} \times \boldsymbol{\xi}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'$$

where $\varepsilon > 0$ is a regularization parameter that effectively controls the minimal size of the vortices.

2.8.2 Vorticity-velocity form

Previously, in Section 2.7, we derived the equations of motion for an incompressible Newtonian fluid, the incompressible Navier-Stokes equations. These equations describe the time evolution of *velocity*. As a gently warm-up to the next section on vortex sheets (Section 2.8.3), we will look at the equations of motion for *vorticity*. As we discussed in the beginning of Section 2.8, working with vorticity is just as valid as working with velocity when considering incompressible fluids. Recall that vorticity is defined as $\boldsymbol{\xi} = \nabla \times \mathbf{u}$. The equations of motion can be obtained by taking the curl of the momentum equation (Equation (2.30)), which gets us

$$\frac{\partial \boldsymbol{\xi}}{\partial t} + (\mathbf{u}\cdot\nabla)\,\boldsymbol{\xi} + (\boldsymbol{\xi}\cdot\nabla)\,\mathbf{u} - \nu\nabla^2\boldsymbol{\xi} = \nabla \times \mathbf{f}. \tag{2.39}$$

The first thing we notice about Equation (2.39) is that the term involving the gradient of pressure has dropped out because the curl of a gradient is zero by vector calculus identity. This has the benefit that we no longer have to solve for the unknown pressure p, however, since velocity still appears in the equation, we have effectively traded solving a scalar Poisson equation for the pressure p for solving a vector Poisson equation for the vector potential $\boldsymbol{\Psi}$. Nevertheless, this may still pay off when vorticity is sparse as we will see in the next section. Other than this fact, Equation (2.39) closely resembles the momentum equation, except for the addition of the new *vortex stretching* term $(\boldsymbol{\xi}\cdot\nabla)\,\boldsymbol{\omega}$.

2.8.3 Vortex sheets

In this next section we will see the equations of motion for a *vortex sheet*, which is simply the concentration of vorticity on a two-dimensional surface M embedded in three-dimensional space $\mathbb{R}^3$. We will also see that these equations bear striking similarity to our surface-based dynamics in Chapter 4.

Physically, a vortex sheet arises when there is a discontinuity in the tangential velocity of two contacting inviscid immiscible fluids with densities ρ_1 and ρ_2 and constant surface tension γ. Letting $(\mathbf{u}_2 - \mathbf{u}_1)$ be the (tangential) velocity discontinuity between the two fluids and $\hat{\mathbf{n}}$ be the unit normal to M pointing into the exterior fluid labeled 2, we can define the *vortex strength* of the sheet M as

$$\boldsymbol{\zeta} \overset{\text{def}}{=} \hat{\mathbf{n}} \times (\mathbf{u}_2 - \mathbf{u}_1)$$

Note that this suggests that $\boldsymbol{\zeta}$ is tangential to M. The equations of motion are obtained as in Wu [1995] or Pozrikidis [2000] and are given by

$$\frac{\mathrm{D}\boldsymbol{\zeta}}{\mathrm{D}t} - (\boldsymbol{\zeta}\cdot\nabla)\,\mathbf{u} + \boldsymbol{\zeta}\,\mathbb{P}(\nabla)\cdot\mathbf{u} = 2A\hat{\mathbf{n}}\times(\overline{\mathbf{a}}-\mathbf{g}) + \frac{4\gamma}{\rho_1+\rho_2}\hat{\mathbf{n}}\times\nabla H \tag{2.40}$$

where $A = \frac{\rho_1-\rho_2}{\rho_1+\rho_2}$ is the Atwood ratio, $\mathbb{P} = \mathbf{I}_3 - \hat{\mathbf{n}}\otimes\hat{\mathbf{n}}$ is the tangential projection operator, $\mathbf{I}_3$ is the 3-by-3 identity matrix and $\overline{\mathbf{a}} = \frac{1}{2}\big(\frac{\mathrm{D}\mathbf{u}_1}{\mathrm{D}t} + \frac{\mathrm{D}\mathbf{u}_2}{\mathrm{D}t}\big)$ is the average of the accelerations on each side of the sheet. Compare this to our surface correction forces in Section 4.4, reproduced in the same notation as Equation (2.40) for convenience

$$\frac{\mathrm{D}\boldsymbol{\zeta}}{\mathrm{D}t} = 2\hat{\mathbf{n}}\times(-\nabla p) + 4\gamma\hat{\mathbf{n}}\times\nabla H$$

where we have set $\beta = 2$ as discussed in Section 4.4. The primary difference to Equation (2.40) is that $(\overline{\mathbf{a}}-\mathbf{g})$ has been replaced with the negative pressure gradient $-\nabla p$. Also, the vortex stretching $-(\boldsymbol{\zeta}\cdot\nabla)\,\mathbf{u}$ and dilation $\boldsymbol{\zeta}\,\mathbb{P}(\nabla)\cdot\mathbf{u}$ terms have been neglected.

Since a vortex sheet is simply the concentration of vorticity, we can compute the velocity induced by the vortex sheet in a similar manner to how we computed the velocity from vorticity in Section 2.8.1. In Chapter 4 we use the Biot-Savart law to recover velocity from the rotational velocities and accelerations in our surface correction algorithm.

2.9 Perfectly Matched Layers

In this section we will describe the theory of perfectly matched layers, which is a technique to absorb radiating waves without reflections. Unfortunately, the theory has gotten a bit of a reputation in computer graphics of being difficult to understand. In part, this is true because most of the theory of perfectly matched layers is found in the computational physics literature and is concerned with Maxwell's equations, not Navier-Stokes. Even the parts of the literature concerned with Navier-Stokes mostly deal with the simpler linearized Euler equations, which are less useful in computer graphics. In this section we try to dispel the myth that the theory of perfectly matched layers is somehow hard to understand and provide some background on the theory readily accessible to computer graphics researchers. The material covered in this section is going to be based loosely on the notes Johnson [2010a] and Johnson [2010b].

2.9.1 Introduction

When solving differential equations numerically by a volumetric discretization as in Section 2.3.4, it becomes necessary to truncate the domain at the grid boundary. Such truncation should be done in a way that does not introduce artifacts into the solution. For differential equations that support traveling waves such as the wave equation (*cf.* Section 2.3.1) or Navier-Stokes, the solutions usually decay too slowly in space and time for it to be practical to wait for "natural" decay and impose Dirichlet or Neumann boundary conditions. The presence of oscillations in the solution also means that any *real* coordinate transformations that maps an infinite domain to a finite one will lead to solutions in the transformed domain that oscillate infinitely fast as we approach the grid boundary and we simply run out of resolution.

In response to this problem Berenger [1994] developed the Perfectly Matched Layer (PML) technique. Instead of using a *real* coordinate transformation, the PML technique works by extending the solution into the complex plane where an oscillating solution is turned into an exponentially decaying one. The ordinary solution is used everywhere except a small layer close to the grid boundary (*cf.* Figure 2.3) where the solution is evaluated in the complex plane. It is possible to achieve an exponentially decaying solution in this layer without perturbing the original solution evaluated at real coordinates (*cf.* Section 2.2), which means that the layer is indeed perfectly matched. It does not matter which boundary condition (Dirichlet or Neumann) is used at the grid boundary. By the time a traveling wave reaches the grid boundary, it will already be exponentially small and so will any unwanted wave reflection off of the grid boundary as a result.

2.9.2 Problem

Assume that we are given some wave-like differential equation of spatial variable $\mathbf{x} \in \mathbb{R}^3$ and time variable $t \in \mathbb{R}$ with solution $\mathbf{u}(\mathbf{x}, t)$ in infinite space. Assume that we are only interested in the solution in some region near the origin $\mathbf{x} = \mathbf{0}$. We would like to truncate the domain outside the region of interest in a way that absorbs all the radiating waves. We are going to use the perfectly matched layers technique to accomplish this.

One of the key assumptions we are going to make is that $\mathbf{u}(\mathbf{x}, t)$ can be written as a superposition of plane waves

$$\mathbf{u}(\mathbf{x}, t) = \int\limits_{\mathbb{R}^n} \widetilde{\mathbf{u}_0}(\mathbf{k}) e^{i(\mathbf{k}\cdot\mathbf{x} - \omega t)} \, \mathrm{d}\mathbf{k}. \tag{2.41}$$

or, in other words, that it comes from an inverse spatio-temporal Fourier transform. In Section 2.3.3 we saw that the kind of differential equations that admit solutions of the form in Equation (2.41) are linear differential equations of the form

$$\frac{\partial \mathbf{u}}{\partial t} = P(D)\,\mathbf{u}$$

where $P(D)$ is skew-adjoint and has constant coefficients. Examples of such equations include the scalar wave equation from Section 2.3.1, Maxwell's equations from electromagnetism, Schrödiger's equation from quantum mechanics, elastic vibrations and many more [Johnson 2010b]. We will discuss the applicability to the incompressible Navier-Stokes in Section 2.9.5.

It is not actually necessary to assume that $\mathbf{u}$ is a superposition of plane waves *everywhere*. It is enough to assume this to be the case *far* from the region of interest (that is, for sufficiently large or small $\mathbf{x}$) where we are going to apply the perfectly matched layer technique.

With the assumptions out of the way, we are ready to apply the perfectly matched layer technique to our problem. The technique proceeds in three steps.

1. Perform an analytically continuation of the solution $\mathbf{u}$ into the complex plane. To be able to do this $\mathbf{u}$ must be analytic, however we have already assumed that $\mathbf{u}$ is a superposition of (analytic) plane waves and a superposition of analytic functions is certainly analytic. Notice that the analytically continued solution $\mathbf{u}$ satisfies the *same* differential equation. This means that the solution is unchanged when evaluated at real coordinates. When it is evaluated along a complex contour, it becomes exponentially damped.

2. Since it is inconvenient to work directly with complex coordinates, we will express the complex coordinates as a function of *real* coordinates. This function will have an imaginary component inside the perfectly matched layer. We then perform a *coordinate transformation* to express the differential equation in real coordinates. More about this in Section 2.9.3.

3. Inside the perfectly matched layer (far from the region of interest near the origin), we now truncate the domain. At this point the solution is already exponentially small and the choice of boundary condition (Dirichlet or Neumann) to apply at the truncated boundary does not matter.

2.9.3 Coordinate transformation

As described in the previous section, it is a bit inconvenient to work directly with complex coordinates. In this section we will show how to express the complex coordinates as a function of *real* coordinates. We will also show how to perform a coordinate transformation to express a differential equation in these real coordinates.

The simplest complex coordinates we can choose are $\widehat{x}(x) = x + i\Sigma(x)$ for some real function Σ that is positive inside the perfectly matched layer and zero everywhere else. Plugging these coordinates into a plane wave (as we did in Section 2.2), we get

$$U_0 e^{i(k\widehat{x}-\omega t)} = U_0 e^{i(kx-\omega t)} e^{-k\Sigma(x)}.$$

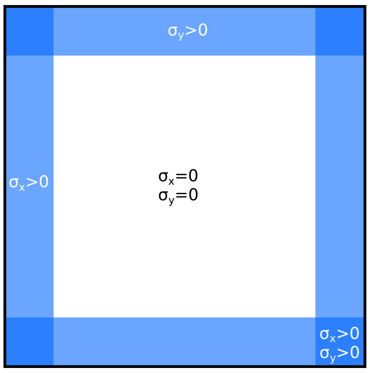


Figure 2.3: Illustration of perfectly matched layers applied to a two-dimensional problem. In the region of interest in the middle, the damping functions $\sigma_x = \sigma_y = 0$ and the solution is unperturbed. Close to the boundary we let $\sigma_x > 0$ or $\sigma_y > 0$ so the solution becomes exponentially damped. Notice that in the corners, both damping functions are positive.

Clearly this is damping if and only if $k\Sigma > 0$, however the amount of damping now depends on k, so higher spatial frequencies will be damped faster than lower frequencies. It would be better if the damping was somehow independent of frequency. We can achieve this by instead using the functions

$$\begin{aligned} \widehat{x}(x) &= x + \frac{i}{\omega}\int^x \sigma_x(x')\,\mathrm{d}x' \\ \widehat{y}(y) &= y + \frac{i}{\omega}\int^y \sigma_y(y')\,\mathrm{d}y' \\ \widehat{z}(z) &= z + \frac{i}{\omega}\int^z \sigma_z(z')\,\mathrm{d}z' \end{aligned} \tag{2.42}$$

where we define the functions σ_x, σ_y and σ_z that are positive inside the perfectly matched layer and zero everywhere else. Plugging these coordinates into a plane wave, we get

$$U_0 e^{i(k\widehat{x}-\omega t)} = U_0 e^{i(kx-\omega t)} e^{-\frac{k}{\omega}\int^x \sigma_x(x')\mathrm{d}x'}.$$

That is, the plane wave is exponentially damped if and only if

$$\frac{k}{\omega}\int^x \sigma_x(x')\,\mathrm{d}x' > 0$$

which is true as long as the *phase velocity* w/k is positive (is travelling to the right) and is in the positive half-plane, or has negative phase velocity (is travelling to the left) and is in the negative half-plane.

In order to apply to apply the coordinate transformation to a differential equation, we need to figure out how the partial derivatives transform. By the chain rule we have

$$\mathrm{d}\mathbf{x} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial \widehat{x}}{\partial x} & \frac{\partial \widehat{x}}{\partial y} & \frac{\partial \widehat{x}}{\partial z} \\ \frac{\partial \widehat{y}}{\partial x} & \frac{\partial \widehat{y}}{\partial y} & \frac{\partial \widehat{y}}{\partial z} \\ \frac{\partial \widehat{z}}{\partial x} & \frac{\partial \widehat{z}}{\partial y} & \frac{\partial \widehat{z}}{\partial z} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial}{\partial \widehat{x}} \\ \frac{\partial}{\partial \widehat{y}} \\ \frac{\partial}{\partial \widehat{z}} \end{bmatrix} = \left(\frac{\mathrm{d}\widehat{\mathbf{x}}}{\mathrm{d}\mathbf{x}}\right)^{-1} \mathrm{d}\widehat{\mathbf{x}}$$

where $\mathrm{d}\widehat{\mathbf{x}}/\mathrm{d}\mathbf{x}$ is the Jacobian matrix. This gives

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+\frac{i}{\omega}\sigma_x(x)} & 0 & 0 \\ 0 & \frac{1}{1+\frac{i}{\omega}\sigma_y(y)} & 0 \\ 0 & 0 & \frac{1}{1+\frac{i}{\omega}\sigma_z(z)} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \widehat{x}} \\ \frac{\partial}{\partial \widehat{y}} \\ \frac{\partial}{\partial \widehat{z}} \end{bmatrix}. \tag{2.43}$$

All that remains is to substitute Equation (2.43) into the differential equation under consideration to complete the coordinate transform. Note, the presence of the angular frequency ω in Equation (2.42) means that we have to make this substitution in *frequency domain* as opposed to time domain. We will see an example of this in Section 2.9.4.

2.9.4 The wave equation

In this section we will show how to apply perfectly matched layers to the wave equation from Section 2.3.1. In one spatial dimension the wave equation (assuming that $c = 1$ for simplicity) is given by

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial t} &= \frac{\partial u}{\partial x} \end{aligned} \tag{2.44}$$

Applying the Fourier transform to the time derivative and the complex coordinate transformation from Equation (2.43) to Equation (2.44), we obtain

$$\begin{aligned} -i\omega\widetilde{u} &= \left(1 + \frac{i}{\omega}\sigma_x\right)^{-1} \frac{\partial \widetilde{v}}{\partial x} \\ -i\omega\widetilde{v} &= \left(1 + \frac{i}{\omega}\sigma_x\right)^{-1} \frac{\partial \widetilde{u}}{\partial x} \end{aligned}$$

which after multiplying both equations by $\left(1 + \frac{i}{\omega}\sigma_x\right)$ and converting back to time domain results in

$$\begin{aligned} \frac{\partial u}{\partial t} + \sigma_x u &= \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial t} + \sigma_x v &= \frac{\partial u}{\partial x} \end{aligned}$$

We note that the only difference to Equation (2.44) is the addition of an exponential damping term in both equations. In two spatial dimensions things get a bit more interesting. Here, the wave equation looks as follows.

$$\begin{aligned}\frac{\partial u}{\partial t} &= \frac{\partial v}{\partial x} + \frac{\partial w}{\partial z}\\ \frac{\partial v}{\partial t} &= \frac{\partial u}{\partial x}\\ \frac{\partial w}{\partial t} &= \frac{\partial u}{\partial y}\end{aligned}$$

We again apply the Fourier transform and complex coordinate transformation to obtain

$$\begin{aligned}-i\omega\widetilde{u} &= \left(1+\frac{i}{\omega}\sigma_x\right)^{-1}\frac{\partial \widetilde{v}}{\partial x} + \left(1+\frac{i}{\omega}\sigma_y\right)^{-1}\frac{\partial \widetilde{w}}{\partial y}\\ -i\omega\widetilde{v} &= \left(1+\frac{i}{\omega}\sigma_x\right)^{-1}\frac{\partial \widetilde{u}}{\partial x}\\ -i\omega\widetilde{w} &= \left(1+\frac{i}{\omega}\sigma_y\right)^{-1}\frac{\partial \widetilde{u}}{\partial y}\end{aligned} \tag{2.45}$$

This time we cannot eliminate the terms involving i/ω so easily. However, if we heuristically define two new variables u_1, u_2 such that $u = u_1 + u_2$ and split the first equation into two as follows

$$\begin{aligned}\frac{\partial u_1}{\partial t} &= \frac{\partial v}{\partial x}\\ \frac{\partial u_2}{\partial t} &= \frac{\partial w}{\partial y}\\ \frac{\partial v}{\partial t} &= \frac{\partial u}{\partial x} = \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial x}\\ \frac{\partial w}{\partial t} &= \frac{\partial u}{\partial y} = \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial y}\end{aligned}$$

we can proceed exactly as in the one-dimensional case and obtain

$$\begin{aligned}\frac{\partial u_1}{\partial t} + \sigma_x u_1 &= \frac{\partial v}{\partial x}\\ \frac{\partial u_2}{\partial t} + \sigma_y u_2 &= \frac{\partial w}{\partial y}\\ \frac{\partial v}{\partial t} + \sigma_x v &= \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial x}\\ \frac{\partial w}{\partial t} + \sigma_y w &= \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial y}\end{aligned}$$

The procedure we just outlined is called the *split* PML for obvious reasons. It is also possible to derive an *unsplit* PML. First multiply the first equation in Equation (2.45) by $\left(1+\frac{i}{\omega}\sigma_x\right)\left(1+\frac{i}{\omega}\sigma_y\right)$, the second equation by $\left(1+\frac{i}{\omega}\sigma_x\right)$ and the third equation by $\left(1+\frac{i}{\omega}\sigma_y\right)$. We get

$$\begin{aligned}
-i\omega\widetilde{u}+\left(\sigma_x+\sigma_y\right)\widetilde{u}+\frac{i}{\omega}\sigma_x\sigma_y\widetilde{u} &= \left(1+\frac{i}{\omega}\sigma_y\right)\frac{\partial\widetilde{v}}{\partial x}+\left(1+\frac{i}{\omega}\sigma_x\right)\frac{\partial\widetilde{w}}{\partial y}\\
-i\omega\widetilde{v}+\sigma_x\widetilde{v} &= \frac{\partial\widetilde{u}}{\partial x}\\
-i\omega\widetilde{w}+\sigma_y\widetilde{w} &= \frac{\partial\widetilde{u}}{\partial y}
\end{aligned}$$

The last two equations are trivial to convert back into time domain. The terms in the first equation that involve i/ω factors correspond to *time integration* in time domain. Collecting all these terms, we obtain

$$-i\omega\widetilde{u}+\left(\sigma_x+\sigma_y\right)\widetilde{u}=\frac{\partial\widetilde{v}}{\partial x}+\frac{\partial\widetilde{w}}{\partial z}+\frac{i}{\omega}\left(\sigma_x\sigma_y\widetilde{u}+\sigma_y\frac{\partial\widetilde{v}}{\partial x}+\sigma_x\frac{\partial\widetilde{w}}{\partial y}\right) \tag{2.46}$$

The trick is to introduce the new variable ψ and the auxiliary equation

$$-i\omega\widetilde{\psi}=\sigma_x\sigma_y\widetilde{u}+\sigma_y\frac{\partial\widetilde{v}}{\partial x}+\sigma_x\frac{\partial\widetilde{w}}{\partial y}$$

which we substitute into Equation (2.46). After converting back into time domain, we end up with the final set of equations

$$\begin{aligned}
\frac{\partial u}{\partial t}+\left(\sigma_x+\sigma_y\right)u &= \frac{\partial v}{\partial x}+\frac{\partial w}{\partial z}+\psi\\
\frac{\partial v}{\partial t}+\sigma_x v &= \frac{\partial u}{\partial x}\\
\frac{\partial w}{\partial t}+\sigma_y w &= \frac{\partial u}{\partial y}\\
\frac{\partial\psi}{\partial t} &= \sigma_x\sigma_y u+\sigma_y\frac{\partial v}{\partial x}+\sigma_x\frac{\partial w}{\partial y}
\end{aligned}$$

In this case we ended up with the same number of variables and equations for both the split and the unsplit PMLs, however, in general the unsplit PML requires more auxiliary variables and equations than the split method. Therefore, we elect to use the split PML in Chapter 5.

2.9.5 Incompressible Navier-Stokes

In this section we briefly discuss the validity of applying the perfectly matched layers technique to the incompressible Navier-Stokes equations from Section 2.7. We reproduce the momentum equation for convenience

$$\frac{\partial\mathbf{u}}{\partial t}+(\mathbf{u}\cdot\nabla)\,\mathbf{u}+\frac{1}{\rho}\nabla p-\nu\nabla^2\mathbf{u}=\mathbf{g}.$$

If we ignore the non-linear advection term $(\mathbf{u}\cdot\nabla)\,\mathbf{u}$ for a moment and assume that the external accelerations are conservative (that is, $\mathbf{g}=\nabla\phi$ for some scalar field

ϕ) the equation does indeed become linear and it can be expressed in the form $\partial \mathbf{u}/\partial t = P(D)\mathbf{u}$ as in Section 2.3.3. To be constant coefficient, however, we must additionally restrict ρ and ν to be constants.

One way to include the advection term and still have a linear equation is to replace it with a term of the form $(\overline{\mathbf{u}} \cdot \nabla)\mathbf{u}$ for some spatially varying $\overline{\mathbf{u}}(\mathbf{x})$. This is what is usually done when time-stepping, where we fix $\overline{\mathbf{u}}$ to be the value $\mathbf{u}^n$ of $\mathbf{u}$ at the beginning of the time step. However, this linearization *still* has spatially-varying coefficients. Strictly speaking, we have to assume that $\overline{\mathbf{u}}$ is *constant*, or at least slowly varying, for the theory to be applicable.

Although the assumptions we have made in this section might seem severe, we emphasize that they must only hold inside the perfectly matched layer (far from the region of interest). We also emphasize that previous work [Hu et al. 2008; Söderström and Museth 2009] violate exactly the same assumptions as we do. The important point is that the perfectly matched layer technique works remarkably well despite this fact.

2.10 Moving surfaces

Each method proposed in this thesis relies on some form of surface representation for a moving surface embedded in three-dimensional space. Moreover, several of the works directly contribute to the state-of-the-art of these surface-based algorithms. For example, in Chapter 3 we show how to establish temporally coherent point-to-point correspondences for incoherent surfaces, and in Chapter 4 we show how to correct errors in liquid surfaces. In this section we give a brief introduction to two of the most popular representations for moving surfaces in computer graphics, representations we will be making heavy use of in this thesis, namely triangle meshes and level sets.

Regardless of which surface representation we choose, there is a basic set of operations that it needs to support for the applications presented later in this thesis. We will go over these in the following. For full generality, we will do this in the smooth setting. There, a dynamic surface can viewed as the image $f(M) = \{f(x) \mid x \in M\}$ of a map $f : M \to \mathbb{R}^3$ from a smooth manifold M into $\mathbb{R}^3$.

- The surface normal $\hat{\mathbf{n}}$ will be useful countless times in this thesis, for instance to compute the surface energy in Chapter 4. We will also need to compute the mean curvature H, which appears in the expression for surface tension used in Chapter 4. In the smooth setting both quantities are easily computed through the formula

$$\Delta f = 2H\hat{\mathbf{n}}$$

where Δ is the Laplace-Beltrami operator (essentially the usual Laplacian confined to the surface), $H : M \to \mathbb{R}$ is mean curvature and $\hat{\mathbf{n}} : M \to S^2$ is the (unit) surface normal (also called the Gauss map) [Crane et al. 2013].

- We will also need to store quantities on the surface. In the smooth setting this is formulated as fields over M, that is, an assignment of some arbitrary quantity ψ to each position on the surface. As an example, a scalar field can be defined as a map $\psi : M \to \mathbb{R}$.
- Next, we want to be able to express an integral over the surface. This again shows up when we define our surface energy in Chapter 4. Letting dA denote the area element, the surface integral of the quantity ψ is
$$\int_M \psi \, \mathrm{d}A.$$
- Finally, we would like to be able to move the surface in an external velocity field $\mathbf{v}(\mathbf{x})$. This shows up in Chapters 3 to 5. In the smooth setting, this is most easily expressed as a differential equation
$$\frac{\partial f}{\partial t} = \mathbf{v}$$

2.10.1 Triangle meshes

A triangle mesh is a triplet $T = (V, E, F)$ of vertices $i \in V$, edges $ij \in E$ and faces $ijk \in F$. These sets can be realized trivially as arrays, though more advanced data structures such as the corner table [Rossignac et al. 2001] or the half-edge data structure [Mäntylä 1987] can be employed to achieve more efficient mesh traversal and updates.

To properly represent a surface embedded in three-dimensional space we need to associate three-dimensional coordinates $\mathbf{x}_i$ to each vertex. Having done so, we can express the surface normal and mean curvature through the Laplace-Beltrami operator applied to the coordinates $\mathbf{x}_i$ by the formula $(\Delta \mathbf{x})_i = 2H\hat{\mathbf{n}}$ as in the smooth case. In the discrete setting the Laplace-Beltrami operator Δ evaluated at a vertex $i \in V$ is given by [Crane et al. 2013]

$$(\Delta\psi)_i = \frac{1}{2} \sum_{j \in \mathcal{N}(i)} \big(\cot\alpha_j + \cot\beta_j\big)\big(\psi_j - \psi_j\big)$$

where $\mathcal{N}(i)$ is the one-ring neighborhood of vertex i and α_j and β_j are the angles across from the edge ij. if the surface is flat we have $H = 0$ and we do not get a normal. In that case it may be better to average the normals of the incident triangles to get a normal at vertex i.

Given an arbitrary quantity ψ_{ijk} discretized per-triangle, the surface integral is given by

$$\sum_{ijk\in T} \psi_{ijk} A_{ijk}$$

where A_{ijk} denotes the area of the given face. Similarly, for a quantity ψ_i discretized per-vertex, the surface integral is given by

$$\sum_{i\in V} \psi_i A_i$$

where A_i denotes the area of the dual cell face or Voronoi region.

Moving a triangle mesh in an external velocity field $\mathbf{v}(\mathbf{x}, t)$ boils down to solving an ordinary differential equation

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}, t). \tag{2.47}$$

Equation (2.47) can be discretized for each mesh vertex $\mathbf{x}_i$ at each discrete time t with simple forward Euler

$$\frac{\mathbf{x}_i^{n+1} - \mathbf{x}_i^n}{\Delta t} = \mathbf{v}(\mathbf{x}, t)$$

as described in Section 2.3.4 or with higher order methods such as fourth-order Runge-Kutta methods [Wojtan et al. 2011].

Naively moving each mesh vertex using this procedure works well if the velocity is simple (*e.g.* if it corresponds to rigid motion) or for small amounts of time, however, if the deformation is more severe it becomes necessary to dynamically improve the mesh by *e.g.* inserting new triangle when existing ones become too stretched, We explain how to achieve this in more detail in Section 3.4.1.

Another complication arises in applications that dictate that surfaces merge or split apart. This happens frequently in physical applications such as liquid simulation. In such cases special care must be taken to detect when such *topology changes* happen and to merge or split the mesh when necessary. Such methods (see Section 3.2 for an overview) usually involve some form of collision detection and mesh surgery. A particularly simple method [Wojtan et al. 2009] that we utilize several times in this thesis compares the triangle mesh to its signed distance function and replaces the mesh with the signed distance reconstruction wherever the two disagree locally. We describe this algorithm in more detail in Section 3.4.3.

2.10.2 The level set method

In the level set method [Osher and Fedkiw 2003] a surface is implicitly defined as the zero level set $\{\mathbf{x} \mid \phi(\mathbf{x}) = 0\}$ of its signed distance function $\phi : \mathbb{R}^3 \to \mathbb{R}$. That is, $|\phi|$ is the distance to the surface with $\phi < 0$ on the inside and $\phi > 0$ on the outside by convention.

In a naive implementation ϕ is sampled on a dense volumetric grid with corresponding volumetric computational and space complexity as a consequence. Modern implementations, such as OpenVDB [Museth 2013], store only a narrow band of voxels around the zero set, which reduces the computational and memory requirements to be proportional to the surface area. This makes makes level sets competitive with triangle meshes.

The normal to the surface is computed as $\hat{\mathbf{n}} = \nabla\phi/||\nabla\phi||$. Since ϕ is a signed distance function it changes with a rate of one unit per unit perpendicular to the surface (*i.e.* in the gradient direction), which means that $||\nabla\phi|| = 1$ so the expression for the normal can be simplified considerably to $\hat{\mathbf{n}} = \nabla\phi$. The mean curvature H is easily computed through the formula

$$2H = \nabla\cdot\hat{\mathbf{n}} = \nabla\cdot\frac{\nabla\phi}{||\nabla\phi||} = \nabla\cdot\nabla\phi = \nabla^2\phi$$

Notice that this mimics the smooth setting.

Let ψ be the value that we are interested in storing "on" the level set. Because the surface is defined implicitly and sampled on a grid, it is not possible to store ψ directly *on* the surface as was the case with triangle meshes. Instead, we store ψ on all grid points near the surface. For this to make sense, we need to ensure that the (volumetric) gradient of ψ is zero in the direction normal to the surface (if ψ was only defined on the surface there could surely not be a gradient in the normal direction)

$$\hat{\mathbf{n}}\cdot\nabla\psi = 0$$

We can achieve this by solving the extrapolation equation until steady state (that is, until $\partial\psi/\partial t = 0$)

$$\frac{\partial\psi}{\partial t} + \hat{\mathbf{n}}\cdot\nabla\psi = 0.$$

A surface integral over a level set is most easily accomplished by integrating over the whole volume against a smoothed delta function δ_ε. We will use

$$\delta_\varepsilon(\phi) = \begin{cases} \frac{1}{2\varepsilon}\left(1+\cos\frac{\pi\phi}{\varepsilon}\right) & \text{if } -\varepsilon \le \phi \le \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

If we now assume that the level set (and its associated quantities) have been discretized onto a grid with uniform grid spacing Δx and that $\mathbf{x}_{ijk} = (i\Delta x, j\Delta x, k\Delta x)$, then the surface integral of ψ is given by

$$\sum_{\mathbf{x}_{ijk}} \psi(\mathbf{x}_{ijk})\delta_\varepsilon(\phi(\mathbf{x}_{ijk}))(\Delta x)^3$$

Moving a surface in the level set can be formulated by applying the chain rule to $\mathrm{d}\phi/\mathrm{d}t$ and noticing the dependence on Equation (2.47)

$$\begin{aligned}\frac{\mathrm{d}\phi(\mathbf{x},t)}{\mathrm{d}t} &= \frac{\partial\phi}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}t} + \frac{\partial\phi}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial\phi}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\partial\phi}{\partial z}\frac{\mathrm{d}z}{\mathrm{d}t} \\ &= \frac{\partial\phi}{\partial t} + \nabla\phi\cdot\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \\ &= \frac{\partial\phi}{\partial t} + \nabla\phi\cdot\mathbf{u} \\ &= \frac{\mathrm{D}\phi}{\mathrm{D}t}\end{aligned} \tag{2.48}$$

where $\mathrm{D}/\mathrm{D}t := \partial/\partial t + (\mathbf{u}\cdot\nabla)$ is the material derivative.

Chapter 3

Tracking Surfaces with Evolving Topology

We present a method for recovering a temporally coherent, deforming triangle mesh with arbitrarily changing topology from an incoherent sequence of static closed surfaces. We solve this problem using the surface geometry alone, without any prior information like surface templates or velocity fields. Our system combines a proven strategy for triangle mesh improvement, a robust multi-resolution non-rigid registration routine, and a reliable technique for changing surface mesh topology. We also introduce a novel topological constraint enforcement algorithm to ensure that the output and input always have similar topology. We apply our technique to a series of diverse input data from video reconstructions, physics simulations, and artistic morphs. The structured output of our algorithm allows us to efficiently track information like colors and displacement maps, recover velocity information, and solve PDEs on the mesh as a post process.

3.1 Introduction

Robust computational representations of deforming surfaces are considered indispensable within many scientific and industrial fields. Medical scientists deduce clues about the human body from the level sets of time-varying voxel data, physicists extract geometric information from simulations and acquisitions of fluid interfaces, and computer graphics professionals generate animations and capture performances in order to entertain audiences. As tools that generate time-evolving surfaces become increasingly commonplace, it is essential that we, as computer graphics researchers, provide better tools for the analysis and computational processing of these forms of animated geometry.

One particular class of evolving surfaces, namely surfaces that change topology through time, is particularly difficult to deal with. Because these surfaces are

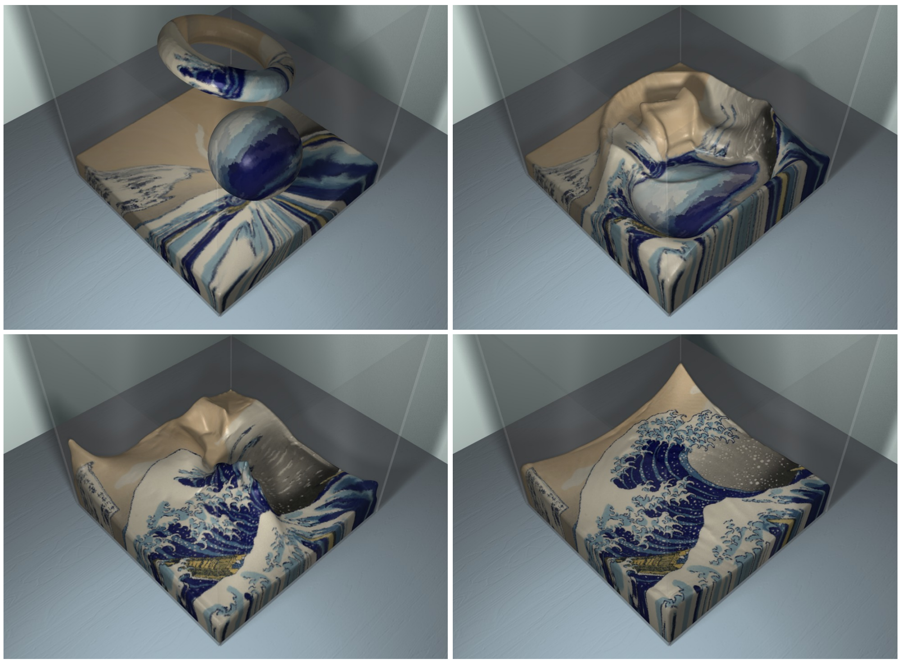


Figure 3.1: Our method recovers a sequence of high-quality, temporally coherent triangle meshes from any sequence of closed surfaces with arbitrarily changing topology. We reliably extract correspondences from a level set and track textures backwards through a fluid simulation.

able to bend, split apart, reconnect themselves, and disappear through time, it is impossible to make any convenient assumptions about their shape and connectivity. For this reason, *implicit surfaces* such as contoured voxel data and metaballs, are extremely popular for representing such time-evolving surfaces. Unfortunately, these implicit surfaces are poorly suited for many important geometric tasks, such as mapping how surface points at one particular time correspond to surface points sometime later.

In this paper, we provide a general, robust method for tracking correspondence information through time for an arbitrary sequence of closed input surfaces. We do not require any context clues such as velocity information or shape priors, and we allow the surfaces to change topology through time. We solve this problem by combining a robust non-rigid registration algorithm, a reliable method for computing topology changes in triangle meshes, and a mesh-improvement routine for guaranteeing numerical accuracy and stability. The output of our method is a series of temporally coherent triangle meshes, as well as an *event list* that tracks how surface vertices correspond through time.

We apply our method to data sets generated by different methods, such as physics simulations using two separate surfacing algorithms, morphing surfaces generated by implicit surfaces, and performance capture data reconstructed from videos. We show that we can reliably extract correspondence information that was absent from the original geometry, and we utilize this information to significantly enhance the input data. Using our algorithm, we are able to preserve important surface features, apply textures and displacement maps, simulate partial differential equations on the surface, and even propagate visual information *backwards* in time. When applied to dynamic shape reconstruction problems, we are able to reliably track the input without making any assumptions about how the data was generated. One can argue that this template-free tracking is an important tool for scientific experiments where it is essential to remove bias from the tools used for information discovery. The contributions of our work are as follows:

- We provide the first comprehensive framework for tracking a series of closed surfaces where topology can change.
- Our algorithm is able to greatly enhance existing datasets with valuable temporal correspondence information. Some examples include displacement mapping of fluid simulations and texture mapping of level set morphs.
- We introduce a novel topology-aware wave simulation algorithm for enhancing the appearance of existing liquid simulations while significantly reducing the noise present in similar approaches.
- Because our method robustly extracts surface information from input data alone, we provide a reliable way to automatically track markerless performance capture data without the need for a template.

3.2 Related Work

Our work is closest to a recent publication of Stam and Schmidt [2011]. They showed that, by examining the input parameters for an implicit surface algorithm, one can derive the surface velocity to create motion blur and more coherent surface animations. By integrating surface velocity through time, they presented a method to approximate point-to-point correspondences which can be used to track texture information. This inspirational work introduced some exciting applications for tracking correspondences through complicated deformations, and we believe that it brought the community a significant step closer to solving the general problem of tracking a topology-evolving surface. Our method is different from theirs in a number of ways. Firstly, we wish to solve the more general problem of tracking an arbitrary input surface sequence, so we do not assume that we know the parameters behind the surface dynamics. Secondly, their correspondence information is only as accurate as their velocity integration, so it is prone to numerical drift. Our method uses a nonlinear shape matching optimization to minimize this drift, and the difference is particularly apparent in the presence of large rotations.

To the best of our knowledge, our method is the first to provide a solution to the problem of registration combined with topology change. For the remainder of this section, we divide the work most related to ours into two camps: those related to deformable shape matching and registration, and those related to surface evolution with topology changes.

Deformable Shape Matching and Registration. The field of dynamic geometry processing is actively involved with the problem of extracting correspondences between inconsistent time-varying meshes [Chang et al. 2010]. Dense and accurate correspondences are critical for temporal shape analysis and surface tracking, making applications such as marker-free human performance capture and shape reconstruction from streams of incomplete 3D data possible. We will focus our discussion on methods that take sequences of meshes or point clouds as input.

Most methods that establish full surface correspondences through time rely on an existing template model or construct it in a separate step. With a *fixed topology* and known geometric state, template models are popular because they simplify the problem of reconstructing geometry and motion. The techniques introduced in [Mitra et al. 2007; Süßmuth et al. 2008] aggregate scan sequences into a 4D space-time surface to build a more complete template. In addition to being limited to fairly small deformations, both methods do not allow the input data to change topology. The statistical framework introduced by Wand et al. [2007] and later improved in [Wand et al. 2009] estimates a globally consistent template model with a fixed topology. While also being restricted to slowly-varying surface deformations, their methods can identify topology variations in the scans. On the other hand, the framework presented in [Li et al. 2009] does use a rough approximation of a template as a prior, preventing wrong topology computations, but focuses on handling deformations that are significantly larger than previous methods using a robust non-rigid registration algorithm. While highly disruptive motions are explicitly treated in the system of Tevs et al. [2012], largely incomplete acquisitions can still damage the template extraction.

Although correspondences are desirable for many geometric analysis and manipulation purposes, a few state-of-the-art reconstruction methods skip the requirement of extracting a template model but aim at simply filling incomplete capture data. The technique presented in [Sharf et al. 2008] is able to produce a watertight surface sequence from extremely noisy input scans using a volumetric incompressible flow prior but suffers from significant flickering in the reconstruction. In the context of fluid capture, Wang et al. [2009] demonstrated a framework to fill holes in partially captured liquid surfaces using a physically guided model. Their method

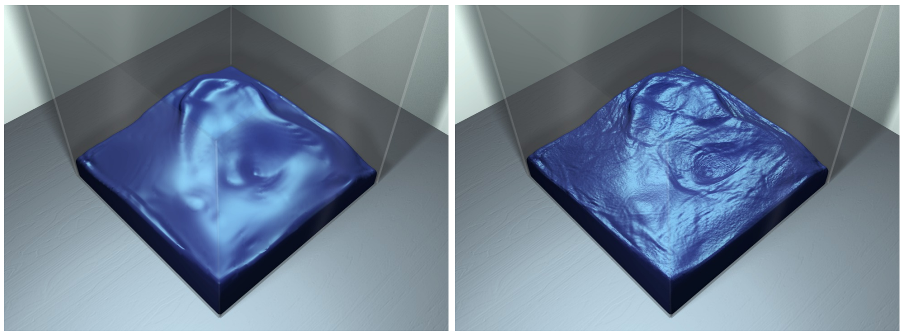


Figure 3.2: Our framework allows us to synthesize high-frequency details using a separate wave simulation (right) on top of a lower resolution pre-simulated fluid surface (left).

achieves time-coherent reconstructions of dynamic surfaces but is restricted to fluid simulations since frame-to-frame correspondences are guided by a simulated velocity field. Recently, Li et al. [2012] demonstrated a shape completion framework for temporally coherent hole filling of incomplete and flickering-affected scans of human performances. Their method makes minimal assumptions about the surface deformation by establishing correspondences within a small time window and thus avoids the more difficult problem of extracting globally consistent correspondences through time.

Conversely, our method is able to establish full correspondences across time-series of input meshes and is not limited to a fixed topology like template-based methods. Our technique is grounded on a general purpose non-rigid registration algorithm similar to [Li et al. 2009; Li et al. 2012] and can therefore be applied widely, ranging from fluid surface dynamics, human body performances, and arbitrary shape morphings.

Surface Evolution with Topology Changes. Several methods exist for tracking topology-changing surfaces through time with the aid of prescribed motions or velocity fields. Level set methods [Osher and Fedkiw 2003] and particle level set methods [Enright et al. 2002] are popular techniques for representing a dynamic implicit surface. These methods consider the zero level set of a voxelized signed distance function, and they integrate velocity information in order to move the function. This integration displaces the zero set of the function, resulting in a moving surface. Müller [2009] used a strategy of repeatedly re-sampling an evolving Lagrangian triangle mesh in order to provide fast surface tracking for fluid surfaces. Semi-Lagrangian contouring [Bargteil, Goktekin, et al. 2006] also utilizes Lagrangian information in the form of extracted surface geometry in order to improve surface tracking. These methods can be used to propagate surface

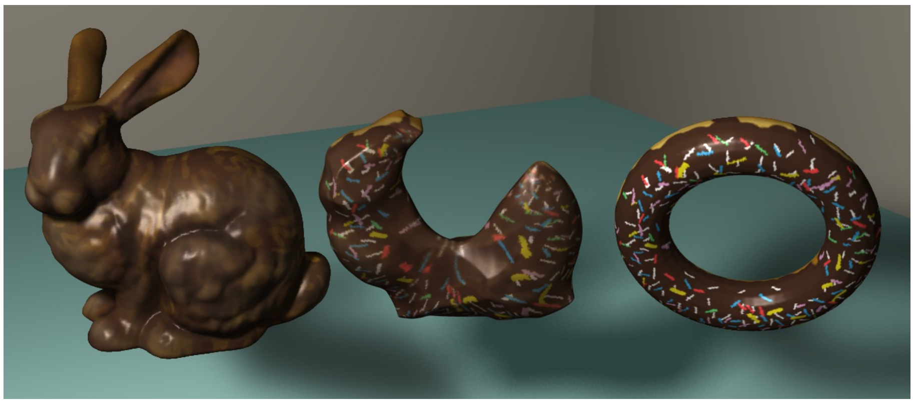


Figure 3.3: A morphing example where surface textures are tracked. Unlike existing techniques, our method does not exhibit ghosting artifacts.

information through time, but they cannot reliably track surface correspondences over large deformations without diffusion because of their strategy of continual re-sampling. Similar to our method, Dinh et al. [Dinh et al. 2005] also tracks texture information on a topology-changing surface. Their method requires the solution of a PDE over space-time, which limits its application to low resolution surfaces over a short amount of time. Our method treats each time step independently, so it is able to handle highly detailed input.

The surface evolvers most similar to ours are mesh-based surface tracking methods [Du et al. 2006; Wojtan et al. 2010; Brochu et al. 2010]. The idea behind these techniques is to evolve a triangle mesh according to a velocity field, which allows for better preservation of geometric features and correspondence information than an implicit surface. These mesh-based methods go hand-in-hand with robust numerical methods for changing mesh topology [Brochu and Bridson 2009; Wojtan et al. 2009; Campen and Kobbelt 2010; Zaharescu et al. 2007; Pons and Boissonnat 2007]. Within our framework, we use a method similar to Wojtan et al. [2009] for changing mesh topology, because of its speed and versatility (Further details are explained in Section 3.4.3).

While each of these works on surface evolution certainly helped inspire ours, we would like to remind the reader that our method solves a significantly different problem of tracking *without any velocity information*. In this light, we do not perceive our method as a competitor to existing fluid simulation techniques, but as a powerful enhancement tool — it allows a user to convert the output from *any simulation type* into a temporally coherent deforming mesh sequence. Our tracked surfaces are a great improvement over implicit surfaces in the information they provide, the details they preserve, and the useful applications that they aid.

3.3 Problem Statement

This paper is concerned with the problem of taking a series of closed surfaces through time as input, and then replacing these surfaces with a sequence of temporally coherent deforming triangle mesh. We wish to allow these input surfaces to have arbitrary shapes and topology, and these shapes and topology are allowed to change significantly from one surface to the next. Because such data can come from a range of diverse sources in practice, we cannot assume any specific domain knowledge, nor can we assume that we are given additional information such as velocity fields. While surface tracking and registration is a widely studied problem, we are unaware of any tracking methods that are both robust to large deformations and arbitrarily complicated topology changes while retaining correspondence information. This is unfortunate, because frequent topology changes result from many common sources such as fluid dynamics, morphing, and erroneous scanned data.

To adequately solve this problem, we must define what it means for two shapes to correspond in the presence of topology changes and find the most appropriate mapping between consecutive pairs of input surfaces. This correspondence information should gracefully propagate through changes in surface topology. We require our method to handle arbitrarily large plastic deformations through time while keeping the computation tractable.

3.4 Method

Our algorithm consists of several interwoven operations: mesh improvement (Section 3.4.1), non-rigid alignment (Section 3.4.2), and topological change (Section 3.4.3). The mesh improvement operation ensures that our output mesh **M** retains high-quality triangles while only minimally re-sampling geometry. The non-rigid alignment step ensures that **M** actually conforms to the desired shapes through time, and the topology change step ensures that the topology of **M** conforms to that of the desired input shapes $\{\mathbf{S}_n\}_{n=1}^{N}$ in each frame. We show that these three operations alone are enough to generate smoothly deforming meshes with high-quality geometry. However, in order to utilize these deforming meshes to their full extent, we also record correspondence information along the way (Section 3.4.4). Finally, we explain how to use the recorded correspondence information to efficiently propagate information forwards and backwards through time as a post-process (Section 3.4.6).

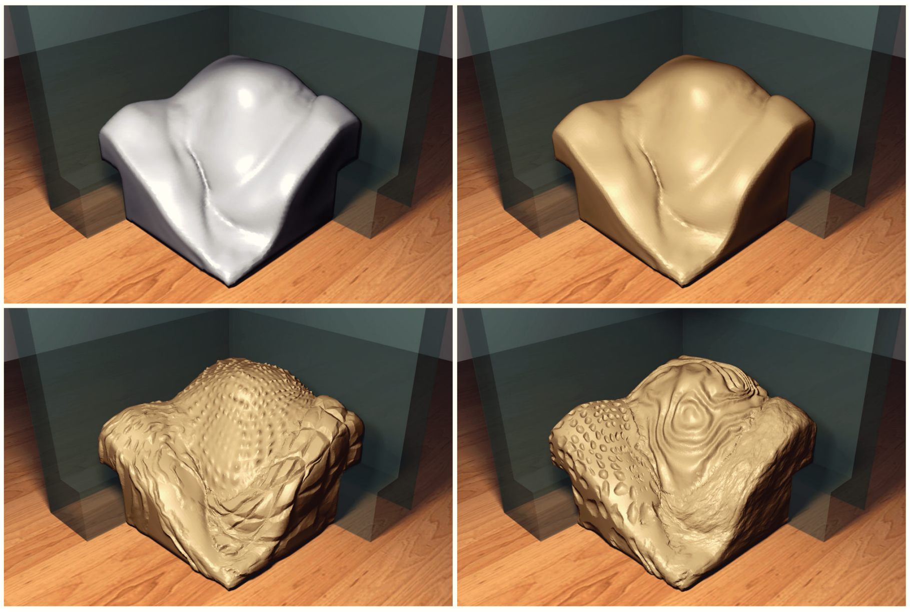


Figure 3.4: Our method can turn a temporally incoherent mesh sequence (upper left) into a coherent one (upper right). We use this tracked mesh to add displacement maps as a post-process without having to re-simulate any physics.

3.4.1 Mesh Improvement

A detailed surface mesh with well-shaped triangles is essential for a wide variety of beneficial computations. In addition to enhancing numerical stability in our non-rigid registration solver (Section 3.4.2) as well as the geometric intersection code in our topology change routine (Section 3.4.3), a triangle mesh free from degeneracies is necessary for such basic operations as interpolation, ray tracing, and collision detection. As we explain later in Section 3.5, the guaranteed mesh quality from our algorithm allows us to densely sample complex textures, generate displacement maps which are less prone to self-intersections, and solve partial differential equations on a deforming mesh using a finite element method.

In our framework, we follow the mesh improvement procedures outlined in the survey by Wojtan et al. [2011]. When edges become too long, we split them in half by adding a new vertex at the midpoint. When edges become too short, or when triangle interior angles or dihedral angles become too small, we perform an edge collapse by replacing an edge with a single vertex. Although we did not implement them in our framework, edge flips are also another excellent mesh re-sampling operation.

When improving a dynamically-deforming mesh, the main challenge is to find the right balance between high-quality triangles and excessive vertex re-sampling. Though we are free to customize these mesh improvement parameters however we

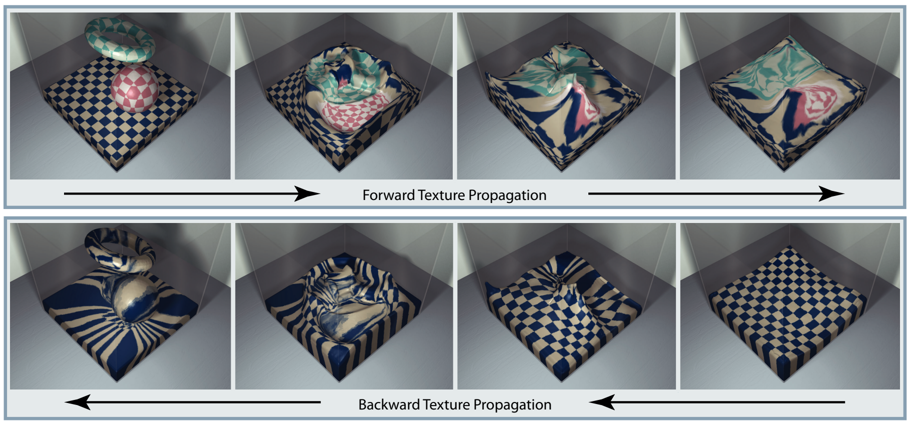


Figure 3.5: These animations show how we can use our algorithm to propagate a texture both forwards and backwards through time. In the bottom animation, the fluid simulation naturally splashes around as it settles into a checker texture.

like, we used similar parameters for all of the examples in this paper. We used a minimum interior angle of 10 degrees, a minimum dihedral angle of 45 degrees, and a maximum:minimum edge length ratio of 4:1. For a more in depth discussion on choosing parameters for these operations, please see [Wojtan et al. 2011].

3.4.2 Non-Rigid Alignment

Our goal is to establish correspondences between a source $\mathbf{M}$ and a target $\mathbf{S}_n$. If we assume that the two shapes have the same topology, we can solve this problem by warping $\mathbf{M}$ onto $\mathbf{S}_n$ while minimizing surface distances and shape distortion. In general, this assumption does not hold, but we may still use non-rigid registration to align the shapes. By simultaneously maximizing geometric similarity and rigidity, surface regions on $\mathbf{M}$ that are compatible with those on $\mathbf{S}_n$ will be aligned, providing dense correspondences within these regions.

We adapt the state-of-the-art bi-resolution registration framework by Li et al. [2009] for non-rigid alignment. Their method is split into two parts to maximize robustness and efficiency: a non-linear optimization that takes care of coarse alignment, and a linearized optimization that aligns fine-scale details. We describe these parts in the following two sections.

Coarse Non-Linear Alignment. Li et al. [2009]'s non-rigid iterative closest point algorithm alternates between estimating correspondences from $\mathbf{M}$ to $\mathbf{S}_n$, and non-rigid deformation of $\mathbf{M}$ that allows correspondences to slide along $\mathbf{S}_n$. Rigidity of the deformation model is relaxed whenever convergence is detected to avoid local minima.

Deformation is achieved using a coarse *deformation graph* that is constructed by uniformly sub-sampling $\mathbf{M}$ such that the distance between deformation graph

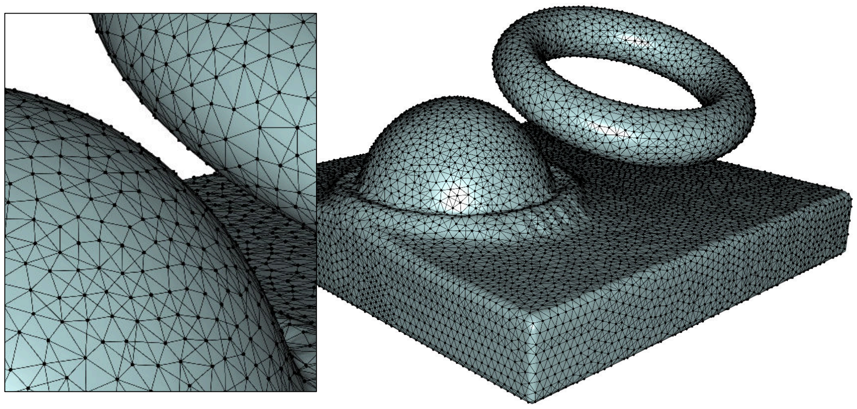


Figure 3.6: Our mesh is augmented with a deformation graph for robust coarse-level non-rigid registration. We use geodesic distances to construct the graph in order to avoid edge connections between surfaces close in Euclidean space but far along geodesics.

nodes is four times larger than the average edge length of $\mathbf{M}$. Instead of computing displacements for each vertex of $\mathbf{M}$, we solve for an affine transformation $(\mathbf{A}_i, \mathbf{b}_i)$ for each graph node. The graph node transformations are transferred to the remaining vertices via linear blend skinning. Letting $\mathcal{N}(i)$ denote the $k = 4$ graph nodes nearest to $\mathbf{x}_i$, we describe the motion of $\mathbf{x}_i$ by a linear combination of the computed graph node transformations. Each $j \in \mathcal{N}(i)$ is weighted as $w_{ij} = (1 - d(\mathbf{x}_i, \mathbf{x}_j)/d_{\text{max}})^2$ and normalized such that $\sum_{j \in \mathcal{N}(i)} w_{ij} = 1$. Here, $d(\cdot, \cdot)$ denotes geodesic distance and d_{max} is the distance to the $(k+1)$th nearest graph node. The choice of using geodesic distances was made to ensure that a vertex is not influenced by graph nodes that are close in Euclidean distance but far along geodesics. This is important, *e.g.* in case of a breaking wave whose tip might be close to the surface in Euclidean but not geodesic distance. Instead of using the same connectivity as the original triangle mesh, a graph edge is formed whenever there exists a vertex in $\mathbf{M}$ influenced by two graph nodes.

When estimating correspondences, the original formulation matches vertices on $\mathbf{M}$ with the closest point on $\mathbf{S}_n$. We instead choose to project a vertex $\mathbf{x}_i$ onto $\mathbf{S}_n$ in the direction of the surface normal to obtain $\mathbf{c}_i$. We have found that this heuristic is significantly better at picking correspondences during large non-rigid deformations, especially where surfaces spread out into thin sheets. To avoid inconsistent alignments, we prune correspondences where the surface normals at $\mathbf{x}_i$ and $\mathbf{c}_i$ are more than 60 degrees apart, or where $\mathbf{c}_i$ is more than three times further from $\mathbf{x}_i$ than the closest point on $\mathbf{S}_n$.

To solve for the affine transformation, we minimize an energy functional that consists of a fitting and some regularization terms. The fitting energy measures how far $\mathbf{M}$ is from $\mathbf{S}_n$ according to the correspondences found above.

$$E_{\text{fit}} = \sum_{i \in \mathcal{V}} (\alpha_{\text{point}} ||\mathbf{x}_i - \mathbf{c}_i||^2 + \alpha_{\text{plane}} \langle \mathbf{n}_i, \mathbf{x}_i - \mathbf{c}_i \rangle^2)$$

Here, $\mathcal{V}$ is the set of deformation graph nodes, $\mathbf{c}_i$ denotes the point on $\mathbf{S}_n$ mapped from $\mathbf{x}_i$ and $\mathbf{n}_i$ denotes the surface normal at $\mathbf{c}_i$. The parameters α_{point} and α_{plane} determine the relative importance of the corresponding point-to-point and point-to-plane energy terms. We use $\alpha_{\text{point}} = 0.1$ and $\alpha_{\text{plane}} = 1$ in all our examples. The larger weight for the point-to-plane term allows the correspondences to slide along $\mathbf{S}_n$ when solving for the deformation, leveraging the coupling between correspondence and deformation optimization.

A second term maximizes the rigidity of the affine transformation, thus minimizing distortion and scaling. This is accomplished by measuring how far $\mathbf{A}_i$ is from a true rotation matrix. Letting $\mathbf{a}_{i1}, \mathbf{a}_{i2}, \mathbf{a}_{i3}$ be the columns of $\mathbf{A}_i$, we obtain

$$\begin{aligned} E_{\text{rigid}} = \sum_{i \in \mathcal{V}} (& \langle \mathbf{a}_{i1}, \mathbf{a}_{i2} \rangle^2 + \langle \mathbf{a}_{i1}, \mathbf{a}_{i3} \rangle^2 + \langle \mathbf{a}_{i2}, \mathbf{a}_{i3} \rangle^2 \\ & + (1 - ||\mathbf{a}_{i1}||)^2 + (1 - ||\mathbf{a}_{i2}||)^2 + (1 - ||\mathbf{a}_{i3}||)^2) \end{aligned}$$

A final term ensures smoothness between edge connected nodes.

$$E_{\text{smooth}} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} ||\mathbf{A}_i(\mathbf{x}_j - \mathbf{x}_i) + \mathbf{x}_i + \mathbf{b}_i - (\mathbf{x}_j + \mathbf{b}_j)||^2$$

The total energy $E_{\text{total}} = \alpha_{\text{fit}} E_{\text{fit}} + \alpha_{\text{reg}} (E_{\text{rigid}} + 0.1 E_{\text{smooth}})$ is minimized using a standard Gauss-Newton solver based on Cholesky decomposition. We alternate between correspondence estimation and surface deformation until convergence, and gradually relax the regularization by dividing α_{reg} by 10. For each $\mathbf{S}_n$ we initialize the optimization with $\alpha_{\text{fit}} = 0.1$ and $\alpha_{\text{reg}} = 1000$.

Fine-Scale Linear Alignment. While the coarse level optimization makes sure that large deformations between $\mathbf{M}$ and $\mathbf{S}_n$ are recovered, a second warping step uses a more efficient (but rotation-sensitive) linear mesh deformation technique to capture high-frequency geometric details in $\mathbf{S}_n$. For each vertex of $\mathbf{M}$, we trace an undirected ray in the normal direction and find the closest intersection point $\mathbf{c}_i$ on $\mathbf{S}_n$.

The optimization uses a point-to-point fitting term $E_{\text{fit}} = \sum_{i \in \mathcal{V}} ||\mathbf{x}_i - \mathbf{c}_i||^2$ and solves for the displacement of each vertex by minimizing the difference between adjacent vertex displacements using $E_{\text{reg}} = \sum_{(i,j) \in \mathcal{E}} |d_i - d_j|^2$.

To avoid self intersections, we prune correspondences that are further than a threshold $\sigma = 0.1$ given a scene bounding box diagonal of 1. Finally, we synthesize fine-scale details from the target on the pre-aligned mesh by minimizing $E_{\text{tot}} = E_{\text{fit}} + E_{\text{reg}}$ using an efficient conjugate gradient solver. Despite the robustness of the proposed non-rigid registration approach, we do not guarantee that every target surface region will have a corresponding source point. Such cases require a change in topology.

3.4.3 Topological Change

This paper considers a more general class of input deformations than most previous methods — we aim to track surfaces that are not only highly deformable, but that may change topology arbitrarily through time. For example, we allow new surface components to appear from nowhere in the middle of an animation, and we expect that entirely disparate surface regions may suddenly merge together. In order to accurately track such extreme behavior in the input data, we build new tools to constrain the topology of our mesh to that of an arbitrary closed input surface.

We base our topology change method on that of Wojtan et al. [2009] with subdivision stitching [Wojtan et al. 2010] as explained in their SIGGRAPH course [Wojtan et al. 2011]. The method takes as input a triangle mesh $\mathbf{M}$ and voxelizes its signed distance function $\phi_{\mathbf{M}}$ onto a volumetric grid. A cubic cell in the volumetric grid is classified as topologically complex if the intersection of $\mathbf{M}$ with the cell is more complex than what can be represented by a marching cubes reconstruction of $\phi_{\mathbf{M}}$ inside the cell. Topologically complex cells are candidates for re-sampling, and triangles of $\mathbf{M}$ inside such cells will be replaced by marching cubes triangles reconstructed from $\phi_{\mathbf{M}}$. This strategy forces $\mathbf{M}$ to change such that its topology matches that of $\phi_{\mathbf{M}}$, effectively making sure that $\mathbf{M}$ changes topology in the event that it intersects itself. See Wojtan et al. [2011] for a detailed exposition.

We chose to use this method primarily because of its flexibility and robustness. We would like the surface to change topology not only when the mesh intersects itself, but also whenever the input geometry happens to change its own topology. Furthermore, because this method is independent of surface velocity, it adds another layer of robustness to our algorithm; in the event that our registration routine produces inaccurate displacement information, the topology algorithm will correct the final shape by drawing new surface geometry directly from $\mathbf{S}_n$.

To do this, we generalize the idea of Wojtan et al.; instead of constraining the topology of the input mesh to match that of its own signed distance function, we constrain the input mesh to match the topology of *any voxelized implicit surface*. We simply voxelize an arbitrary implicit surface Θ, and replace the signed distance function $\phi_{\mathbf{M}}$ in the original with our new function Θ. The algorithm then compares the topology of the mesh $\mathbf{M}$ to the topology of Θ, and replaces $\mathbf{M}$'s triangles with triangles from the extracted isosurface of Θ wherever $\mathbf{M}$ and Θ have a different local topology. We can refer to this generalized topology change routine as ConstrainTopology($\mathbf{M}, \Theta$). Using this terminology, the original algorithm of Wojtan et al. can be executed by calling ConstrainTopology($\mathbf{M}$, $\phi_{\mathbf{M}}$).

Within our deformation framework, we use this generalized topology change algorithm in two ways: first to ensure that the deforming mesh changes topology if it intersects itself, and second, to ensure that the deforming mesh has the same topology as the target input data. These actions can be computed by calling ConstrainTopology($\mathbf{M}$, $\phi_{\mathbf{M}}$) and ConstrainTopology($\mathbf{M}$, $\phi_{\mathbf{S}_n}$), respectively, where $\mathbf{S}_n$ is the target mesh from the input data. We will specify the exact order in which to call these functions in section Section 3.4.5.

3.4.4 Recording Correspondence Information

Throughout the computation of our deforming mesh $\mathbf{M}$, we want to track how its correspondences evolve through time. The previously mentioned mesh modification routines can cause significant changes in correspondence information, and we must track how these changes occur.

The mesh deformation algorithm described in Section 3.4.2 is Lagrangian in nature, so it moves individual vertices to their new locations at each frame in the animation sequence. Consequently, the vast majority of vertex locations in our mesh at a given frame number correspond exactly to the location of that same vertex at earlier and later frame numbers. For these vertices, information about their corresponding position at different points in the sequence is implicit; vertex i in frame number $n-1$ corresponds exactly with i in frame n.

The only vertices which do *not* have this trivial correspondence with vertices in different frames are the few vertices which were created or destroyed due to re-sampling. Within our framework, the only way to create new vertices is via topological change (Section 3.4.3) or edge and triangle subdivision (Section 3.4.1). The only way for us to destroy vertices is via topological change (Section 3.4.3) or edge collapse (Section 3.4.1). Note that some other potential mesh improvement procedures like mesh fairing [Jiao 2007; Brochu and Bridson 2009; Stam and Schmidt 2011] improve triangle quality at the expense of re-sampling correspondence information by diffusing it along the surface. For this reason, we did not use such fairing procedures in Section 3.4.1.

For each transition between two frames, we track these re-sampling events (edge subdivision, triangle subdivision, edge collapse, topology change) in what we call an *event list*. The event list stores detailed information about each re-sampling event, and it is sorted by the order in which the re-sampling events took place. Each event in the list records information of the form $(\mathcal{V}_{\text{in}}, \mathcal{V}_{\text{out}}, f(\mathcal{V}_{\text{in}}), b(\mathcal{V}_{\text{out}}))$, where $\mathcal{V}_{\text{in}}$ is a set of the input vertices, $\mathcal{V}_{\text{out}}$ is a set of the output vertices, $f(\mathcal{V}_{\text{in}})$ is a function that assigns information to $\mathcal{V}_{\text{out}}$ as a function of $\mathcal{V}_{\text{in}}$, in case we want to propagate information forwards. Similarly, $b(\mathcal{V}_{\text{out}})$ is a function that assigns information to $\mathcal{V}_{\text{in}}$ as a function of $\mathcal{V}_{\text{out}}$, in case we want to propagate information backwards.

When we subdivide an edge (i, j) between two vertices i, j, a new vertex k is created on the line connecting i and j. In this case, the event list records $(\{i, j\}, \{i, j, k\}, k \mapsto (i, \alpha, j, 1 - \alpha), \emptyset)$, where α denotes the barycentric coordinate of k. Notice that we omit the trivial correspondences $i \mapsto (i, 1)$ and $j \mapsto (j, 1)$. Similarly, when we subdivide a triangle (i, j, k) by adding a new vertex l somewhere inside the triangle, we record $(\{i, j, k\}, \{i, j, k, l\}, l \mapsto (i, \alpha_i, j, \alpha_j, k, \alpha_k), \emptyset)$. As before, $\alpha_i, \alpha_j, \alpha_k$ denote barycentric coordinates. Finally, when we collapse an edge (i, j) and replace the two endpoints i, j by a new vertex k at the barycenter of i and j, we record $(\{i, j\}, \{k\}, k \mapsto (i, 0.5, j, 0.5), \{i \mapsto (k, 1), j \mapsto (k, 1)\})$.

When a topological change occurs, surfaces can split wide open and entire patches of new geometry can be created. For each patch of new geometry after the topology change, we propagate information from the vertices on the boundary of the patch inward, using a breadth-first graph marching algorithm (similar to Yu et al. [2012]). Though several propagation strategies are valid at this point (during the marching algorithm, each new vertex could simply copy information from its nearest neighbor, it could distribute information evenly throughout the patch, e.g. by solving an elliptic PDE, etc.), we chose a strategy of each vertex taking the average of the information from its visited neighbors during the breadth-first march. For each new vertex that is created, our event list records the list of boundary vertices, the new vertex, and the linear combination of boundary vertices that results from this marching and averaging. There is no backward correspondence assignment for these vertices.

Lastly, vertices can be deleted in a topological merge. We treat such operations the same way that we treat new vertices that result from a topology change, but in reverse: before the patch of vertices is destroyed, we march inward from the boundary of the patch of deleted vertices and propagate information using the same averaged vertex scheme. For each vertex that is deleted, our event list records the list of boundary vertices, the new vertex, a null forward operation, and the linear combination of boundary vertices that results from the marching and averaging operation.

3.4.5 Summary of the Tracking Algorithm

We review the steps of our tracking method in Algorithm 3.1. Our method begins by initializing a triangle mesh $\mathbf{M}$ to the first frame of the input mesh sequence $\{\mathbf{S}_n\}_{n=1}^{N}$. We then immediately call our mesh improvement routine (Section 3.4.1) to ensure that $\mathbf{M}$ consists of high-quality geometry. Next, we enter the main loop of our algorithm, which visits each of the input meshes $\mathbf{S}_n$ in turn. In each iteration

Algorithm 3.1 Pseudocode for our topology changing surface tracker.

```
1: Mesh $\mathbf{M}$ = LoadTargetMesh($\mathbf{S}_1$)
2: ImproveMesh($\mathbf{M}$)
3: for frame $n = 2 \to N$ do
4:     LoadTargetMesh($\mathbf{S}_n$)
5:     CoarseNonRigidAlignment($\mathbf{M}$, $\mathbf{S}_n$)
6:     FineLinearAlignment($\mathbf{M}$, $\mathbf{S}_n$)
7:     ImproveMesh($\mathbf{M}$)
8:     $\phi_{\mathbf{M}} \leftarrow$ CalculateSignedDistance($\mathbf{M}$)
9:     ConstrainTopology($\mathbf{M}$, $\phi_{\mathbf{M}}$)
10:    $\phi_{\mathbf{S}_n} \leftarrow$ CalculateSignedDistance($\mathbf{S}_n$)
11:    ConstrainTopology($\mathbf{M}$, $\phi_{\mathbf{S}_n}$)
12:    ImproveMesh($\mathbf{M}$)
13:    SaveEventListToDisk($n$)
14:    SaveMeshToDisk($\mathbf{M}$)
15: end for
```

we use our course non-rigid alignment routine (Section 3.4.2) to align the low-resolution features of **M** as closely as possible with those of $\mathbf{S}_n$. Once the coarse alignment has terminated, we perform a fine-scale linearized alignment in order to ensure that all of the high-resolution details of **M** line up with $\mathbf{S}_n$. At this point in the algorithm, we have deformed our mesh **M** such that it lines up with the input data frame $\mathbf{S}_n$. This deformation may cause the triangles of **M** to stretch and compress arbitrarily, so we again perform a mesh improvement in order to clean up the overly deformed elements.

Next, we must account for the fact that **M** may have self-intersections. We execute the basic topology change algorithm in Section 3.4.3 by first computing a voxelized signed distance function near the surface of **M** and then ensuring that **M** has the same topology as the zero isosurface of this function. This step mainly cleans up any large self-intersections in the mesh by merging surface patches together. Next, we execute a topology change algorithm again, but this time we constrain **M** to match the topology of $\mathbf{S}_n$. This step ensures that we split apart any surfaces in **M** which stretches over gaps in $\mathbf{S}_n$, as well as merge any separate regions of **M** that are actually merged in $\mathbf{S}_n$. The extra topology constraint additionally acts as a fail-safe by re-sampling parts of **M** in the rare event that the alignment algorithm was unable to find correspondences for all of **M**.

At this point in the algorithm, our mesh **M** can consist of triangles with arbitrarily poor aspect ratios, because the topological sewing algorithm only cares about the connectivity of the mesh and not the condition of the individual mesh elements. We therefore call our mesh improvement routine once again to ensure that the mesh is fit for another round of tracking. Note that throughout this entire algorithm, we document any re-sampling operations that occur (potentially in lines

2, 7, 9, 11, and 12 of Algorithm 3.1) and add them to our event list (Section 3.4.4). In the final two steps of this loop, we save our event list and the mesh **M** itself to disk. We then start the loop again with the next frame of animation $\mathbf{S}_{n+1}$.

3.4.6 Propagating Information as a Post-Process

After we have finished tracking the input geometry (after all of the steps in Section 3.4.5 have run until completion), we have a series of temporally coherent animation frames of a mesh **M** that deforms and changes topology. Furthermore, we also have a per-frame event list that describes exactly how correspondences propagate throughout the animation. We can use this list to pass information like surface texture and surface velocity from one frame to the next. To pass information forward in time, we run through the event list in the order that each event took place, and, using the notation from section Section 3.4.4, we pass information to re-sampled vertices using the function $f(\mathcal{V}_{\text{in}})$. Similarly, we pass information backwards in time by running backwards through the event list and using $b(\mathcal{V}_{\text{out}})$.

3.5 Applications

Having detailed our method for obtaining a temporally coherent parameterization of an arbitrary sequence of closed manifold meshes (Section 3.4), we shift our focus to applications. We show how we can apply our method to track a broad range of different incoherent surfaces and how we can exploit extracted correspondence information to significantly enhance the meshes in a variety of different ways.

Displacement Maps. Displacement maps provide an efficient way of adding geometric detail to an animation as a post-process, avoiding costly re-simulation or geometry acquisition. We recover a temporally coherent mesh sequence from a physically-based Eulerian viscoelastic simulation [Goktekin et al. 2004] with a periodically re-sampling surface tracker similar to [Müller 2009] (Figure 3.4). Our method faithfully conforms to the target shape in every frame with minimal re-sampling.

To showcase our temporally coherent parameterization and high mesh quality, we apply two different displacement maps to the mesh sequence. We represent a displacement map as a per-vertex scalar designating the normal direction displacement of each vertex. Using our data structure (Section 3.4.6), we propagate displacements applied in the first frame to all later frames. Compared to tracking, propagation is almost instantaneous, taking only a few seconds for the entire animation. Swapping in a different displacement map is thus fast and effortless.

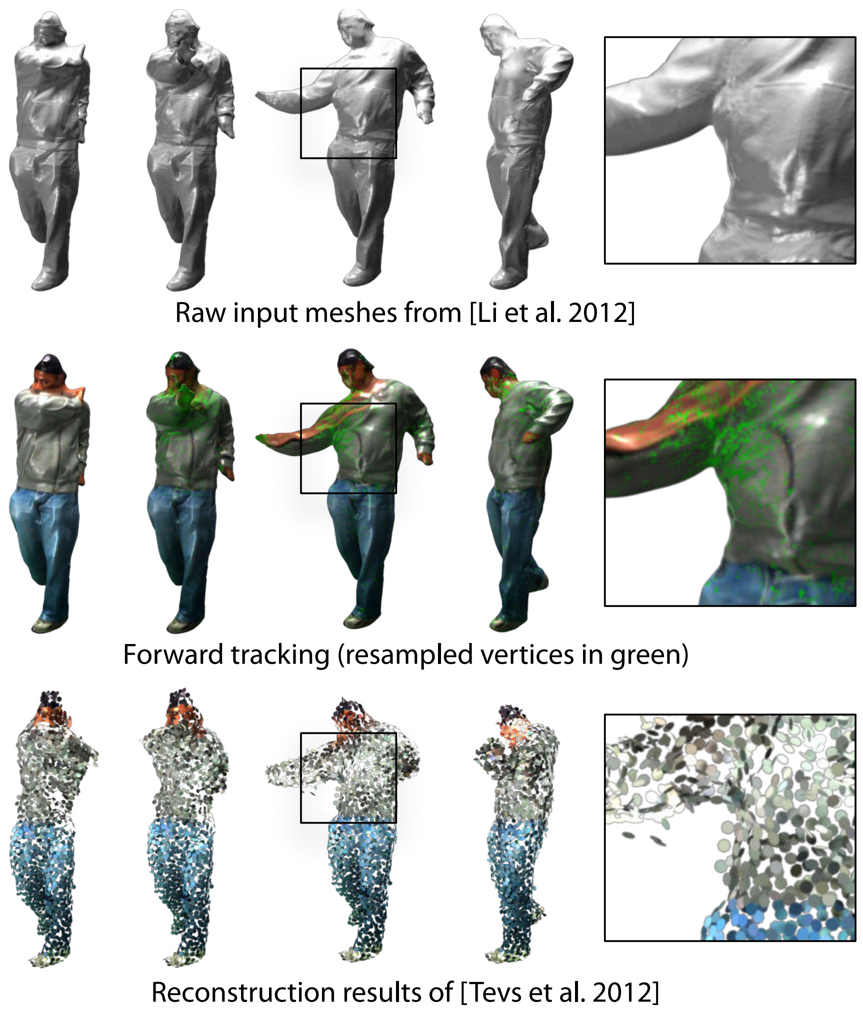

Raw input meshes from [Li et al. 2012]
Forward tracking (resampled vertices in green)
Reconstruction results of [Tevs et al. 2012]

Figure 3.7: Top: Input performance capture data has inconsistent vertices across frames and exhibits topological variations. Middle: Our method seamlessly handles topology changes and ensures high-quality triangles. Resampled vertices from our mesh improvement algorithm are marked in green. Bottom: The method of Tevs et al. (visualized as a point cloud) is prohibitively expensive for long, detailed mesh sequences and fails to capture the correct motion.

Compare this to the state of the art without our method, where an animator instead would have to re-run the entire simulation to change the geometry.

Color. It is often useful to texture implicit surfaces in production [Sumner et al. 2003; Wiebe and Houston 2004]. Because of the large computational costs of liquid simulation, it is particularly convenient to add detail to a lower resolution simulation as a post-process, *e.g.* by applying foam or deep water textures. Figure 3.5 shows a splashy liquid scene which comes from a standard Eulerian solver using the Level Set Method [Osher and Fedkiw 2003] to track the free-surface. We track an incoherent sequence of marching cubes reconstructions of the level sets from the simulation.

Similar to displacement maps, we propagate colors applied in the first frame to all later frames. Our accompanying video shows a checkerboard pattern and a lava texture propagated through time. Further exploiting our temporal data structure, we propagate colors applied in the last frame *backwards* in time to the first frame (Figure 3.5). This technique allows us to enhance the splashy animation with an interesting artistic expression where an image is slowly revealed as the dynamics settle (Figure 3.1).

Wave simulation. Texture is one way of enhancing a low resolution liquid simulation, however, correct computation of light transport for effects like caustics is easier with real geometry. Our method is not limited to static displacement maps (mentioned above), but allows for procedural displacements as well. In particular we may improve the fidelity of the splashy liquid simulating mentioned above (Figure 3.2) by adding a dynamic displacement map. Because our method yields particularly high-quality surface triangles with minimal re-sampling, we are able to use the resulting mesh to solve partial differential equations. Inspired by recent fluid animation research [Thürey et al. 2010; Yu et al. 2012], we augment our surfaces with a time-varying displacement map, computed as the solution to a second order wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h. \tag{3.1}$$

Here, h is wave displacement in the normal direction, ∇^2 is the discrete Laplace operator computed with cotangent weights [Botsch et al. 2010], and c is a user-chosen wave speed. We use our event list to transfer the state variables (wave heights h and velocities in the normal direction v) from one frame to the next, and we integrate the system using symplectic Euler integration with several sub-cycled time steps per input frame. One may optionally choose to add artificial damping to the simulation for artistic reasons by multiplying h by a $(1-\varepsilon)$ factor in each step. No artificial damping was used in our simulations.

Our wave simulation method is novel in that it retains tight control over wave energy sources. We only add wave heights precisely at the locations in space-time where topological changes occur. This stands in opposition to previous work, which recomputes wave heights every time step based on surface geometry. The result of this distinction is that our simulations are much less likely to introduce energy due to numerical errors. Our simulations have a dramatically high signal-to-noise ratio – we can clearly see interesting wave interference patterns persist throughout the entire simulation.

Morph. Another application of our method is transferring colors through morphs that change topology between arbitrary genera (Figure 3.3). We use a simple linear blend between signed distance functions to create the morph, and we subsequently obtain a coherent mesh by tracking it with our framework. We start by propagating color backwards from the final frame, and then we use the colors which were propagated to the first frame to obtain a base texture. In this way an artist can see where important feature points end up on the target shape to aid in creating a more natural morph. To obtain the morph in Figure 3.3, we additionally blend between the two forward and backward propagated colors.

Performance Capture. Performance capture has numerous applications such as video games and filmmaking. Due to noise and occlusion, captured data often exhibits non-physical topology changes. Unlike previous methods, we are able track captured data with topology changes while obtaining temporally coherent correspondences (Figure 3.7). We apply a texture in the first frame and propagate it forward. Regions that are unoccluded throughout the sequence are tracked faithfully.

3.6 Evaluation

We performed an extensive series of tests to evaluate our method. We used the viscoelastic simulation (Figure 3.4) as a testbed while we varied parameters, turned off various parts of our code, and attempted alternative approaches. Please see our accompanying video for visualizations of these tests.

In Figure 3.9, we show how our method compares to the naïve approach of simply projecting the tracked mesh $\mathbf{M}$ onto the input $\mathbf{S}_n$. Tangential drift is severe even in the case of simple translation. Next, we compare our method to one without fine-scale registration (line 6 of Algorithm Algorithm 3.1). Since the graph-based registration works on a coarse scale and only influences vertices in $\mathbf{M}$ through linear blend weights, this modified method is unable to correctly register small features. Such errors accumulate over time, causing a rough, lumpy surface that ignores the

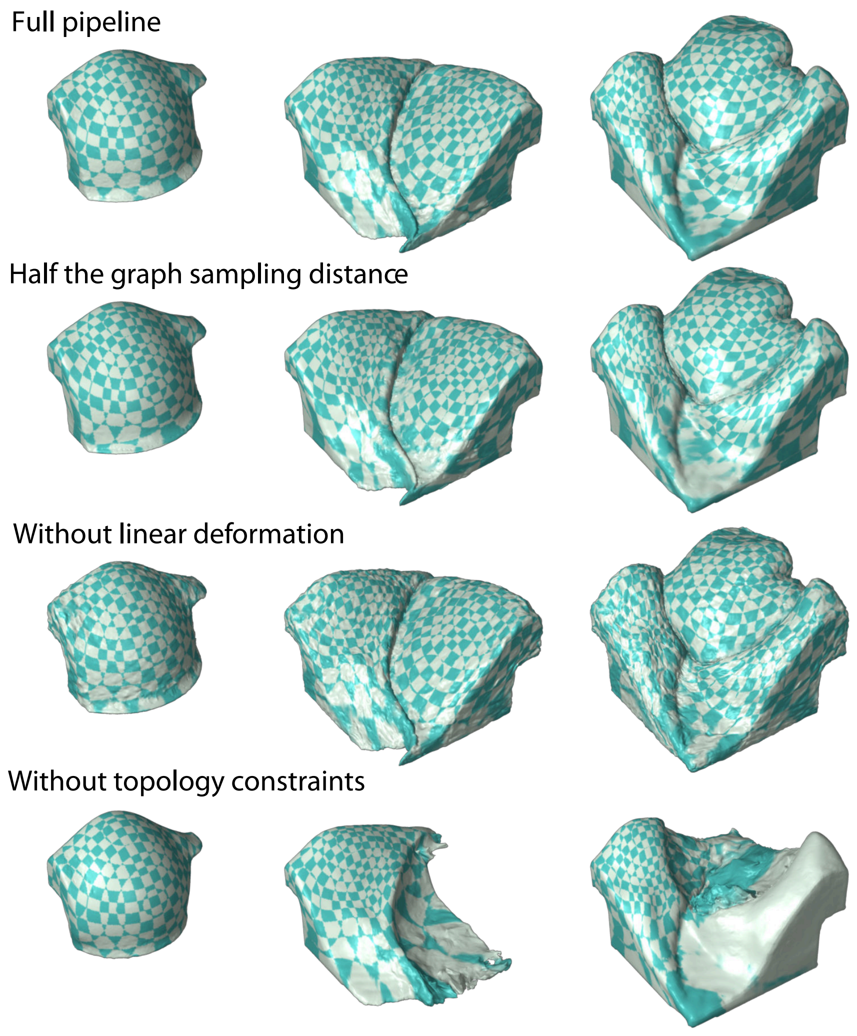

Full pipeline
Half the graph sampling distance
Without linear deformation
Without topology constraints

Figure 3.8: Comparison between our full pipeline and leaving out individual stages of our surface tracking framework.

fine-scale details of the input. Our full algorithm clearly does not exhibit these problems, showing why the fine-scale optimization of Section 3.4.2 is necessary.

Our tests also show that the topology constraint (Section 3.4.3, line 11 of Algorithm 3.1) is essential for robust tracking. The tests in our video illustrate how a method without this constraint is unable to cope with drastic changes in input topology. An obvious example in the viscoelastic simulation is the sudden introduction of new components in later frames — when the topology constraint is turned off, the non-rigid registration algorithm was unable to recognize these components without manually creating a template. Another important feature of the topology constraint is that it acts as a convenient failsafe. Should the registration routine fail to fully conform to the target shape, the topology constraint fills in regions

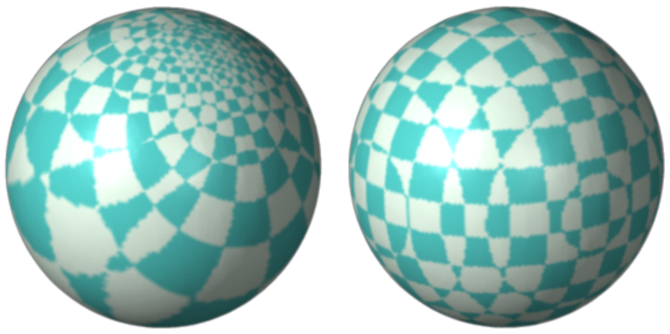


Figure 3.9: The difference between projection (left) and our non-rigid registration technique (right). Simple projection causes severe distortion of the surface, while our registration reliably provides accurate correspondences.

of mismatched geometry. As a result, our full algorithm is quite robust to poor parameter choices for the alignment, and poor alignment only leads to additional re-sampling (as opposed to an unrecoverable failure).

Within a given frame, time complexity of our method is dominated by coarse non-rigid alignment (Section 3.4.2, line 5 of Algorithm 3.1). Sampling density of the deformation graph is the most critical parameter to the time complexity of the non-rigid alignment, since it dictates the number of variables in the non-linear optimization problem. Sensitivity of our algorithm to different sampling densities is examined in the supplementary video. In addition to the density used in Figure 3.4, we also ran the algorithm with both half- and quarter-sampling density. The video shows that reduced sampling densities lead to increased re-sampling, but the result remains similar to our high-quality tracking. Conveniently, this allows use of a lowered sampling density to get a fast approximation of the algorithm's output before committing to solving with a high sampling density.

Another way to reduce the time complexity of our method is to use sparser input. We experimented with five, ten and twenty-five times sparser input than the results shown in Figure 3.4. As seen in the video, our method is robust to sparse input and produces reasonable correspondences, even for the example where we use only sixteen out of the original 400 frames in the input for tracking.

The memory complexity of our algorithm is similarly dominated by the non-rigid alignment. However, because we only do pairwise alignment, memory consumption is independent of the length of the sequence of input data. In other words the space complexity scales with the number of vertices in **M**.

We have gathered statistics for all of our application examples. We summarize these results in Table 3.1. All measurements were performed on a standard PC with an Intel i7-2600K processor and 16 GB of memory. We note that our implementation has not been optimized for performance and is mostly sequential.

	ViscElast	Splash	Morph	PerfCap
Vertices	60k-300k	280k-380k	77k-96k	214k-369k
Frames	400	500	100	111
Frame time	45-153s	105-220s	17-21s	150-174s
Coarse reg.	87-93%	81-89%	67-73%	70-73%
Fine reg.	3-8%	11-18%	19-23%	24-27%

Table 3.1: Summary of statistics for our topology changing surface tracker. Time spent on mesh improvement and topology changes is negligible compared to alignment and is omitted in the table. Timings exclude file I/O operations.

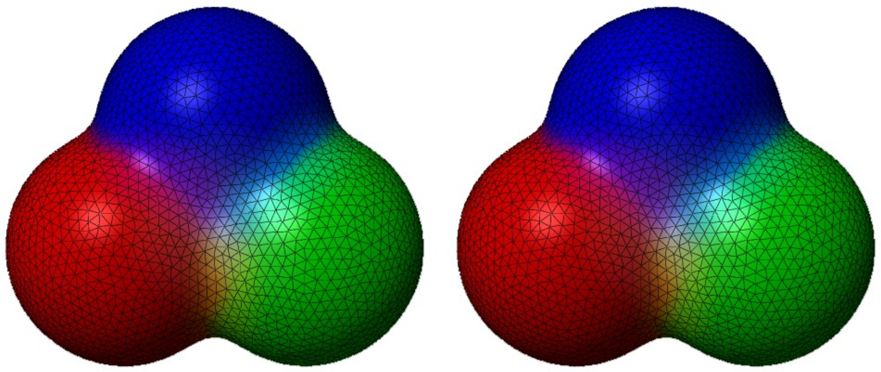


Figure 3.10: Stam and Schmidt introduced this shape as a benchmark for evaluating the accuracy of an implicit surface tracking algorithm. After one complete rotation, our algorithm's output (right) is virtually identical to the analytical solution (left).

Comparison to other methods. As detailed in Section 3.2, the method of Stam and Schmidt [2011] is significantly different from ours. While this is an admittedly biased comparison, we show how our method performs with their example of three blended blobs rotating about the origin (see Figure Figure 3.10). Our algorithm explicitly solves for the globally most rigid deformation, so we obtain practically perfect tracking whereas Stam *et al.* show slight tangential drift and color diffusion. We imagine their problem would be exacerbated with larger time steps, while ours remains accurate.

3.7 Discussion

Since our tracking approach is sequential and does not rely on higher level deformation priors, we do not guarantee drift free tracking. For purposes such as tracking extended performance capture recordings, dynamic body shape statistics and elastic deformation models could be incorporated to prevent accumulations of tracking errors. More generally one could exploit the temporal mesh sequence by combining the result of forwards and backwards tracking [Kim, Liu, Llamas, and

Rossignac 2007; Kagaya et al. 2011]. Nevertheless, none of our examples, including the performance capture example, exhibited any noticeable drift when propagating the texture from the first frame to the end despite the drastic topology variations and large deformations in the input data. Therefore, we did not further investigate these temporal schemes.

Our temporally coherent meshes retain high-quality elements and low-valence vertices, even in the presence of highly non-rigid deformations and topology changes, as can be seen in our supplementary video. Our bound on triangle and dihedral angles in particular make sure that high-valence vertices and skinny triangles are avoided. Since we do not currently implement edge flips, we resample vertices more often than we would otherwise, especially when the surface is compressing or stretching. As our results show, this does not turn our to be a big problem in practice, however, we would like to implement edge flips in the future.

Our method is meant to find surface correspondences through arbitrary deformations while remaining faithful to the input motion. When given a severely stretched deformation as input, an exactly tracked set of surface correspondences will inevitably exhibit severe stretching as well. In situations such as this, a minimally distorted mapping through time is actually incorrect behavior as far as our algorithm is concerned. Our strategy of matching geometry while minimizing tangential drift unavoidably causes distortion when propagating visual data such as texture, as can be seen in *e.g.* Figure 3.5. Our method does, however, allow the user to relax the energy term that punishes tangential drift, thereby giving some control over distortion. If specific requirements are sought, such as maximal conformality of textures, one would have to tailor our method for that particular use-case. We view this as an interesting direction for future work. Texture synthesis [Bargteil, Sin, et al. 2006; Kwatra et al. 2007] is but one possible solution to the challenging problem of texture stretching.

Because our method is based on shape matching, we are unable to track surfaces invariant under our energy functions; a surface with no significant geometric features (like a rotating sphere) will not be tracked accurately. However, it would be easy to augment our method with additional priors such as velocity information in order to handle such featureless cases.

The biggest limitation of our method is the fact that we are currently limited to closed manifold surfaces due to the algorithm we use for performing topology changes. This method assumes that for any arbitrary point in space we must unambiguously decide whether it is inside or outside the surface.

3.8 Conclusion

We have presented a novel approach that takes a sequence of arbitrary closed surfaces and produces as output a temporally coherent sequence of meshes augmented with vertex correspondences. The output of our algorithm is useful for a variety of applications such as (dynamic) displacement maps, texture propagation, template-free tracking and morphs. We have also demonstrated the robustness of the method to parameters as well as input. In the future we would like to extend the method to handle non-closed surfaces, as well as explore problem-specific applications of our general-purpose framework.

Chapter 4

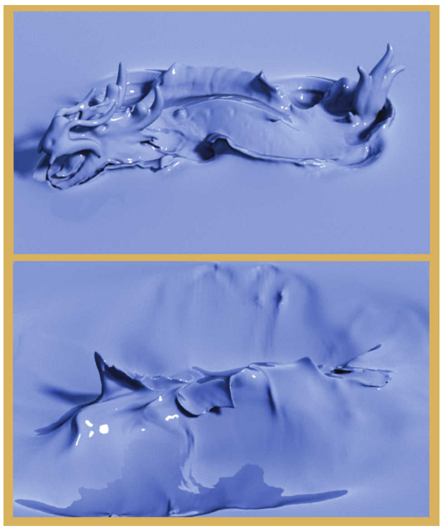
Liquid Surface Tracking with Error Compensation

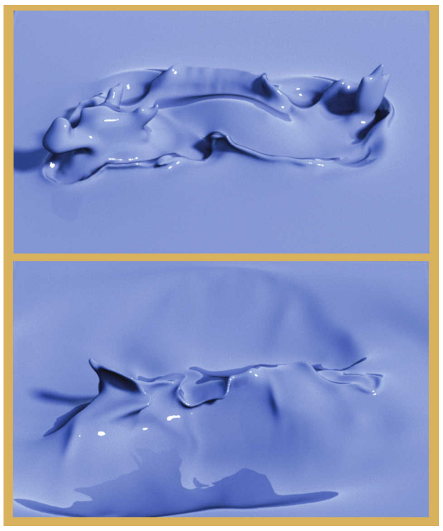
Our work concerns the combination of an Eulerian liquid simulation with a high-resolution surface tracker (e.g. the level set method or a Lagrangian triangle mesh). The naive application of a high-resolution surface tracker to a low-resolution velocity field can produce many visually disturbing physical and topological artifacts that limit their use in practice. We address these problems by defining an error function which compares the current state of the surface tracker to the set of physically valid surface states. By reducing this error with a gradient descent technique, we introduce a novel physics-based surface fairing method. Similarly, by treating this error function as a potential energy, we derive a new surface correction force that mimics the vortex sheet equations. We demonstrate our results with both level set and mesh-based surface trackers.

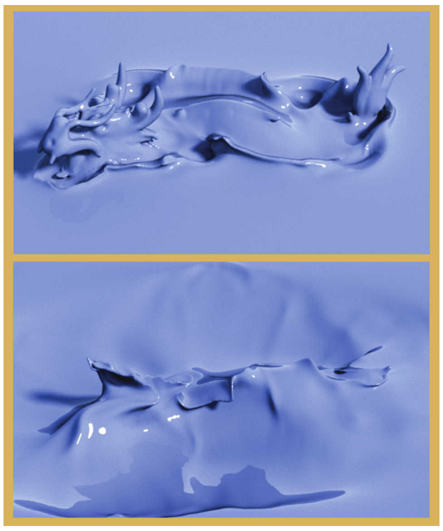
4.1 Detailed surface tracking

This paper addresses the problem of tracking a liquid surface in an Eulerian fluid simulation. Within the field of computer graphics, Eulerian fluid simulation has become commonplace, with standard methods relying on a rectilinear grid or tetrahedral mesh for solving the Navier-Stokes equations [Bridson 2008]. The problem becomes significantly more complicated when we wish to simulate a free surface, such as when animating liquid. Correct treatment of this free surface requires special boundary conditions as well as some additional computational machinery called a *surface tracker*, such as the level set method [Osher and Fedkiw 2003] or a moving triangle mesh [Wojtan et al. 2011].

When animating a free surface, almost all of the visual detail is directly dependent on this surface tracker, because the surface is often the only visible part of the resulting fluid simulation. In order to make a simulation as detailed and visually


(a) Original simulation


(b) Smoothed


(c) Our smoothing

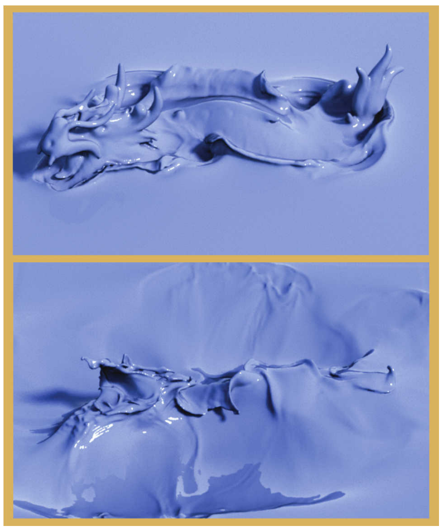

(d) Our dynamics

Figure 4.1: Our method permits high-resolution tracking of a low-resolution fluid simulation, without any visual or topological artifacts. The original simulation (a) exhibits sharp details and low-resolution banding artifacts. Smoothing the surface tracker (b) hides the artifacts but corrodes important surface features. We propose a smoothing technique (c) that preserves sharp details while selectively removing surface tracking artifacts, and a force generation method (d) that removes visual artifacts with strategically placed surface waves. Our algorithms are general and apply to both level sets as well as mesh-based surface tracking techniques.

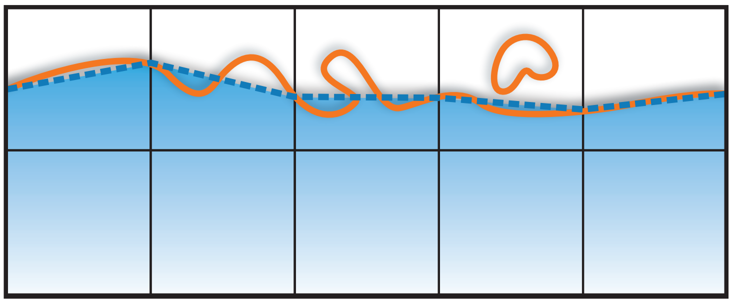


Figure 4.2: If the surface tracker (orange) is much more detailed than the simulation grid (black squares), then the simulation can only work with a rough approximation of the surface (blue). The mismatch between the orange and blue surfaces can create visual artifacts like permanent surface kinks and floating droplets.

rich as possible, we must add detail to the surface tracker. The computational cost of solving the Navier-Stokes equations scales with the volume of the simulation. Therefore, adding details to the surface by simply increasing the number of computational elements quickly becomes intractable. The problem can be somewhat alleviated by speeding up computational bottlenecks like the pressure projection step [Lentine et al. 2010; McAdams et al. 2010], but ultimately the volumetric complexity remains an obstacle. On the other hand, the costs of surface tracking only scales with the surface area, so the immediate temptation here is to increase the resolution of the surface tracker while keeping the fluid simulation resolution fixed. This strategy of only increasing the surface resolution has produced some beautiful results in the past [Goktekin et al. 2004; Bargteil, Goktekin, et al. 2006; Heo and Ko 2010; Kim et al. 2009; Wojtan et al. 2009], but it introduces visual and topological errors that limit its usefulness with extremely detailed surfaces (Figure 4.2).

To see where these errors come from, we consider the relationship between the surface tracker and the fluid simulation. While the surface tracker certainly acts as the source of visual detail, it is also responsible for communicating the location of the free surface to the fluid simulation. The fluid simulation then converts the shape of this free surface into Dirichlet boundary conditions for a Poisson equation. After solving this Poisson equation, the fluid simulation then adds pressure forces to ensure that any subtle variations near the free surface are accounted for in a manner consistent with the Navier-Stokes equations. However, a problem occurs if we lose information when conveying the free surface shape from the surface tracker to the fluid simulation; if the surface tracker is significantly more detailed than the fluid simulation, then there is no way to adequately encode all of the subtleties of the free surface into the boundary conditions. As a result of these mismatched levels of detail, the fluid simulation cannot recognize highly detailed surface features, and it cannot supply the necessary high-resolution pressure forces. Consequently, high resolution surface structures will clearly violate natural fluid motion, because they ignore the pressure term of the Navier-Stokes equations — the fluid simulation simply does not have enough degrees of freedom to prevent unphysical states in the surface tracker.

Previous methods have either ignored these errors, applied surface smoothing, or added additional detail to the fluid simulation in order to address these problems. While surface smoothing eventually removes unphysical high-resolution details, it also removes important physically valid motions, and it has no physical basis (or is based on unphysically strong and over-damped surface tension). Refining the fluid simulation detail near the surface is certainly a valid strategy, but perfectly matching the resolution of a detailed surface tracker often comes with significant extra implementation effort and computational overhead.

Our paper presents a fundamentally different approach for reconciling the difference between a high resolution surface tracker and a low resolution velocity field. We first propose a novel error metric that identifies and quantifies any unphysical surface behaviors by contrasting the current state of the surface tracker with the set of physically valid surface states. Once we have this information, we can use the gradient of this error function in a couple of different ways. We first introduce a novel physics-based surface fairing method that quickly removes surface artifacts while preserving physically-valid surface details. Next, we derive a novel surface correction force that removes artifacts with strategically placed gravity and surface tension waves. We show that this approach conveniently mimics the vortex sheet form of the Navier-Stokes equations while seamlessly integrating into an Eulerian simulation.

The contributions of our paper are as follows:

- Theoretical insight into the problem of coupling a high resolution surface tracker to a low resolution fluid simulation.
- A novel error metric for quantifying the physical validity of a fluid surface tracker.
- A surface fairing algorithm for fluid-like surfaces that clearly out-performs standard smoothing techniques.
- A surface correction force that removes high-resolution artifacts while preserving physically-valid details.
- Applications to both level set and mesh-based surface trackers.

4.2 Previous Work

Our algorithm concerns the combination of an Eulerian discretization of the Navier-Stokes equations with free-surface boundary conditions. The book by Bridson [2008] provides an excellent overview of the common fluid simulation methods in the field of computer animation. In order to simulate a free surface (for example, at the boundary of a liquid), Dirichlet boundary conditions must be enforced by specifying an exact value for the fluid pressure. First order boundary conditions simply set the pressure at the center of a boundary cell, but this strategy leads to aliasing artifacts due to the assumption that the fluid boundary precisely lines up with the simulation grid. Significantly higher accuracy can be achieved by using second order "ghost fluid" boundary conditions [Enright et al. 2003], which use linear extrapolation to set the surface pressure.

Several methods exist for representing and tracking a moving liquid surface. The most common Eulerian surface tracking method for fluid simulation is the level set method [Osher and Fedkiw 2003], which maintains a signed distance function and implicitly represents the surface wherever the function is equal to zero. The volume of fluid method [Hirt and Nichols 1981] also uses an Eulerian strategy to track the surface; by explicitly tracking the amount of fluid in each cell, the boundary can be observed by locating fractionally-filled fluid cells. Lagrangian surface tracking methods use particles or meshes [Wojtan et al. 2011] to explicitly represent the surface. Whereas topology changes are implicit for Eulerian surface tracking routines, they must be carefully computed for Lagrangian mesh-based methods [Brochu and Bridson 2009; Wojtan et al. 2009]. Hybrid surface tracking techniques, like the particle level set method [Enright et al. 2002] attempt to combine the merits of both Lagrangian and Eulerian surface tracking techniques.

High resolution surface, low resolution grid. As mentioned above, many researchers seek to extract additional richness from a simulation by significantly increasing the amount of detail in the surface tracker while fixing the resolution of the underlying fluid simulation. Goktekin et al. [2004] used a high-resolution particle level set to track a low-resolution viscoelastic fluid simulator. Similarly, Wojtan and Turk [2008] and Wojtan et al. [2009] used a Lagrangian mesh to retain detail at a much higher resolution than a viscoelastic simulation. Bargteil, Goktekin, et al. [2006] used an octree and Heo and Ko [2010] used a pseudo-spectral method to maximize the resolution of an Eulerian surface tracker for a fixed fluid simulation.

The main drawback with intentionally mismatching the surface and simulator resolutions is that the surface tracker tends to retain details that are invisible to the simulation [Brochu et al. 2010]. This is less of a problem for a viscoelastic motion, because unphysically-retained surface features may resemble rigid and elastic behavior. Eulerian surface representations, particularly with semi-Lagrangian advection, will naturally lose detail over time; so an overly-detailed tracker can help make up for this loss in detail, although they will preserve visual artifacts if they are too successful. See Figure 4.5 for an example of these visual artifacts.

Removing artifacts. One strategy for eliminating the artifacts caused by a high-resolution surface tracker is to adaptively increase the fluid simulation resolution near the boundary. Losasso et al. [2004], Hong and Kim [2005] and Kim, Liu, Llamas, Jiao, et al. [2007] used an octree to adapt a fluid simulation to a high-resolution level set. Brochu et al. [2010] introduced a simulator based on an adaptive Voronoi diagram in order to match the resolution of a high resolution mesh surface. The general strategy of adding detail to the fluid simulation will naturally remove artifacts due to mismatched resolutions, but it can be computationally expensive and handling spatial adaptivity may introduce further simulation artifacts.

Instead of adapting the fluid simulation to the surface tracker, several methods try to make the high-resolution surface conform to the low-resolution physics. Wojtan and Turk [2008], and Kim et al. [2009] attempt to remove high-frequency visual artifacts using smoothing algorithms, while Yu and Turk [2010] use anisotropic smoothing kernels to bias the loss of surface detail. Williams [2008] and Thürey et al. [2010] attempt to make up for volume-loss artifacts with bi-Laplacian smoothing. While such smoothing approaches may be effective in small doses, they will destroy many interesting surface details when applied with too much enthusiasm, and they do not produce physically correct surface motions. Smoothing in this manner is related to over-damped surface tension, which may be appropriate for small-scale viscous flows but is inaccurate for inviscid liquid phenomena.

In an attempt to make up for the lack of detailed surface motions when combining a low-resolution simulation and a detailed surface, Thürey et al. [2010], Bojsen-Hansen [2011] and Yu et al. [2012] propose using high-resolution dynamic surface waves. These methods mask high resolution surface artifacts with rippling motions, but they are based on inappropriate restrictions such as shallow water, height-field, and constant wave speed assumptions. These restrictions also require that the surface tracker must be homeomorphic to a low-resolution simulation boundary in order to function properly, while our method naturally removes topological inconsistencies with Rayleigh-Taylor-like instabilities. The method of Wojtan et al. [2010] removes topological inconsistencies in the surface tracker by re-sampling the surface, but it does not address the problem of removing surface noise.

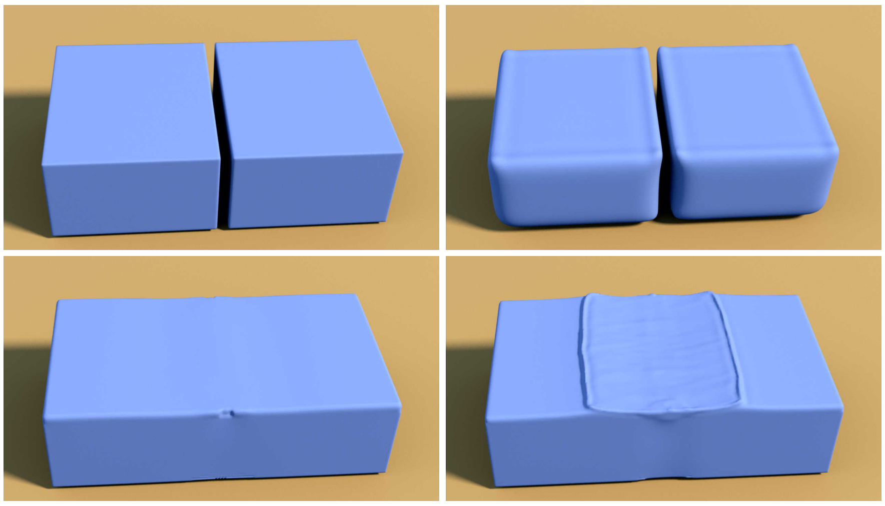


Figure 4.3: This instructive example inserts a high-resolution cut in the surface of a low-resolution simulation. The cut is smaller than a grid cell, so the original fluid simulation (top left) ignores it. Smoothing the surface tracker (top right) leads to extreme loss of volume and surface details. Our smoothing algorithm (bottom left) quickly fills in the gap, and our dynamics algorithm (bottom right) converts the artifact into fluid energy.

Vortex methods. Our method results in equations that resemble the vortex sheet equations. Several researchers have used vortex particle methods [Selle et al. 2005; Park and Kim 2005; Pfaff et al. 2009; Kim et al. 2012] and vortex sheet methods [Pfaff et al. 2012; Brochu et al. 2012] to add details and generate turbulence in fluid simulations, though these methods do not directly address the difficulties of simulating high-resolution motion with a free-surface. The method of Kim et al. [2009] uses a high-resolution surface tracker coupled with the vortex sheet equations to drive a low-resolution vorticity confinement force. This strategy serves to add interesting low-resolution turbulence to the simulation, but it also enhances high-resolution surface noise. Our method only adds turbulence in areas that exhibit unphysical behavior and uses it to remove surface noise artifacts.

4.3 Method

As mentioned in Section 4.1, our main source of unphysical behavior is the conversion from the detailed surface tracker into pressure boundary conditions. We first aim to quantify these errors.

4.3.1 In the absence of Surface Tension

We observe that, in an analytical solution to the Navier-Stokes equations in the absence of surface tension, the pressure at the free surface is equal to that of the surrounding air. Following standard practices for simulating liquids, we assume that the air pressure is a constant zero along the interface [Bridson 2008]. Next, we note that the zero level set of pressure perfectly coincides with the location of the free surface. Because the gradient of a function is always orthogonal to its level sets, we infer that the pressure gradient should be perpendicular to the free surface. Accounting for the fact that the pressure is positive inside the liquid, we can further state that the pressure gradient must be anti-parallel to the surface normal in order for a flow to be consistent with the Navier-Stokes equations (Figure 4.4a). We use this information to define an energy function:

$$E_0 = \int_{\partial\Omega} \mathbf{n} \cdot \nabla p \; dA \tag{4.1}$$

where $\mathbf{n}$ is the surface normal at a point on the surface, ∇p is the pressure gradient, dA is the area of a surface tracking element, and $\partial\Omega$ is the entire free surface according to the high resolution surface tracker. Intuitively, the energy is minimized when the free surface is physically valid (when the normal is anti-parallel with the pressure gradient). We evaluate the surface normal from the surface tracker and the pressure gradient from the fluid simulation, so any deviation from the minimum energy state encodes an unphysical disagreement between the surface tracker and the fluid simulation.

We propose to reduce this error by following the direction of steepest descent of the energy function. The energy gradient is the derivative of Equation (4.1) with respect to its free variables. In our case, the surface tracker is overly-detailed and under-constrained, so we will only adjust the control variables of the surface tracker. We approximate divergence-free motion by constraining our surface tracker adjustments to be local rotations; thus, our degrees of freedom are the orientations of the surface tracker normals $\mathbf{n}$.

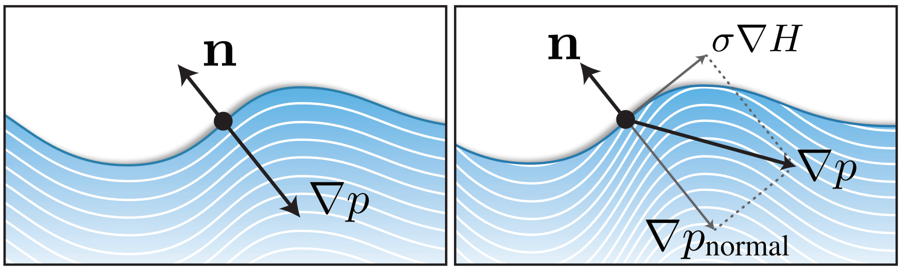
Taking the partial derivative of Equation (4.1) with respect to $\mathbf{n}$, the direction of steepest descent for surface normal is:

$$-\frac{\partial E_0}{\partial \mathbf{n}} = -\nabla p \; dA. \tag{4.2}$$

However, the normals must remain unit length, so we reformulate the energy gradient as a local rotation and arrive at the equation:

$$\mathbf{n} \times \left(-\frac{\partial E_0}{\partial \mathbf{n}}\right) = \nabla p \times \mathbf{n} dA, \tag{4.3}$$

which encodes the area-weighted rotational velocity and axis of rotation as a vector magnitude and direction respectively. This equation tells us that we should rotate the normal away from the pressure gradient, with a strength proportional to the magnitude of the pressure force, if we wish to decrease the surface error.



(a) without surface tension (b) with surface tension

Figure 4.4: A schematic of the simulated liquid (blue) and its level sets of constant pressure (white lines). The surface normal **n** is perpendicular to the fluid surface, while the pressure gradient ∇p is perpendicular to the surfaces of constant pressure. In the presence of surface tension, the tangential component of ∇p is proportional to the gradient of mean curvature.

4.3.2 Including Surface Tension

For large scale flows, surface tension forces are practically negligible, and the above analysis is sufficient. For smaller scale flows, however, we must incorporate effects due to surface tension.

In the presence of surface tension, the pressure at the free surface is $p = \sigma H$, where σ is the surface tension coefficient and H is the mean curvature of the free surface at that point. This pressure can vary along the surface, so the pressure gradient will have a tangential component (Figure 4.4b). As a result, we can no longer assume that the pressure gradient is normal to the free surface. However, the tangential variation of the pressure is fully defined by our free surface boundary conditions, so the tangential component of the pressure gradient is equal to the gradient of the surface pressure: $\nabla p_{\text{tangent}} = \sigma \nabla H$. We decompose the pressure gradient into normal and tangential parts and solve for the normal component:

$$\nabla p_{\text{normal}} = \nabla p - \sigma \nabla H \tag{4.4}$$

Using the same reasoning as in the previous section, the normal component of this pressure gradient should be anti-parallel to the surface normal in a fully resolved fluid simulation. Our surface tracker is more detailed than the fluid simulation and the surface normals will vary, so we introduce the following energy function:

$$E_{ST} = \int_{\partial\Omega} \mathbf{n} \cdot (\nabla p - \sigma \nabla H)\, dA \tag{4.5}$$

which is again minimized when the surface tracker represents a physically valid configuration. We compute the partial derivative of Equation (4.5) with respect to the surface normals:

$$-\frac{\partial E_{ST}}{\partial \mathbf{n}} = (-\nabla p + 2\sigma \nabla H)\, dA \tag{4.6}$$

This derivative is slightly more complicated, because H depends on $\mathbf{n}$, as explained in the Appendix. We then encode the result as a rotation

$$\mathbf{n} \times \left(-\frac{\partial E_{ST}}{\partial \mathbf{n}}\right) = \nabla p \times \mathbf{n}\, dA + 2\sigma \mathbf{n} \times \nabla H dA \tag{4.7}$$

The first term of this update rule is identical to the case without surface tension. The second term indicates that we should rotate the surface normal towards the direction of increasing curvature if we want surface tension to smooth out the surface.

As before, we evaluate the pressure gradient from the fluid simulation and we evaluate the normals from the surface tracker. We are left with a choice of whether to evaluate the mean curvature H using the fluid simulation or the surface tracker. We first note that small values of H will be computed correctly regardless of where we evaluate it, because low curvatures are easily represented on low-resolution fluid grids. However, large values of H cannot be accurately computed on the low-resolution simulation grid and will be clamped to some maximum value related to its Nyquist limit. As a result, high-resolution surface tension motions will be ignored if we compute H using the low resolution fluid simulation. On the other hand, computing H from the surface tracker will allow high-resolution motions, but it will not necessarily give us the same value of ∇p_{normal}. However, the second term of Equation (4.7) will always act to reduce the surface curvature, so we can be confident that it will at least move in the right direction until H is reduced and its computation becomes consistent with the fluid simulation. For this reason, we choose to evaluate H on the surface tracker.

4.4 Applications

Now that we have defined error functions and their gradients, we propose a few different strategies for reducing unphysical behaviors in the surface tracker.

4.4.1 Physics-based surface fairing

Our first application uses the above analysis to derive a gradient-descent technique for reducing unphysical behavior. The gradients derived in the previous section are weighted by area, so we first divide the equation by the area of the surface element to get a local rotational motion. Then, by assigning local rotations to each point on the surface, we present a physically-based surface fairing rule for fluid simulations:

$$\omega = \alpha(\nabla p \times \mathbf{n} + 2\sigma \mathbf{n} \times \nabla H) \tag{4.8}$$

where ω represents the angular velocity at a point on the surface tracker and α is a user-tunable smoothing parameter. In our implementation, we convert

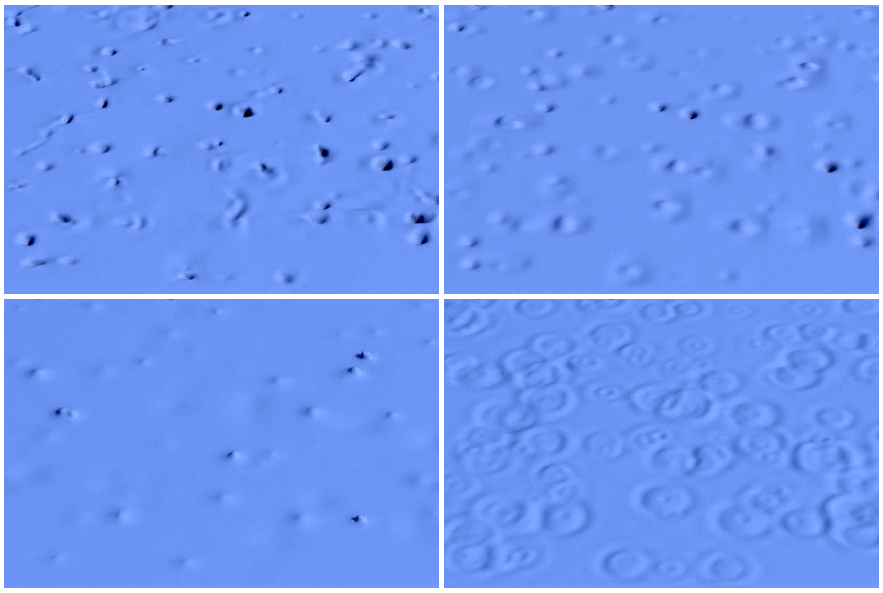


Figure 4.5: The original simulation (top left) cannot remove high resolution noise. After many iterations, Laplacian smoothing (top right) slowly diffuses errors across the surface. Our smoothing method (bottom left) immediately targets and flattens artifacts, and our dynamics algorithm (bottom right) converts artifacts into waves.

this rotational velocity ω into a local high-resolution velocity field using a finite-kernel approximation to the Biot-Savart law. We provide implementation details in Section 4.5.

When applied steadily throughout the progress of the fluid simulation, this procedure has the remarkable effect that it filters the surface at a rate proportional to the magnitude of the pressure gradient — nearly static liquid surfaces are quickly smoothed out while ballistic motions are left untouched (Figure 4.8). Note that this update has no effect if the surface tracker is the same resolution as the fluid simulation; in this case, the normal component of the pressure gradient will precisely line up with the surface normal and the effect of our update rule will disappear. This procedure only acts to correct errors due to a mis-match between the surface tracker and the boundary conditions of the fluid simulation, and the effect smoothly fades away as the fluid simulation accuracy increases.

4.4.2 Fluid simulation on the surface tracker

Another option is to utilize Equation (4.5) as a physical potential energy. We derive surface forces from the gradient of this energy and factor out the per-element area to get an equation for angular acceleration. The result is essentially the time derivative of our surface fairing algorithm:

$$\dot{\omega} = \beta(\nabla p \times \mathbf{n} + 2\sigma \mathbf{n} \times \nabla H) \tag{4.9}$$

where $\dot{\omega}$ is the angular acceleration at some point on the surface tracker, and β is a user-tunable parameter analogous to a spring constant or squared wave speed. Instead of simply smoothing out errors in the surface tracker, this approach transforms surface artifacts into water waves (Figure 4.5).

We note that our error correction forces are remarkably similar to the buoyancy and surface tension terms of the vortex sheet equations [Pozrikidis 2000], hinting that a good choice for our tuning parameter β is a value of twice the Atwood ratio:

$$\beta \approx 2(\rho_{\text{liquid}} - \rho_{\text{air}})/(\rho_{\text{liquid}} + \rho_{\text{air}}) = 2$$

The main differences between our method and previous vortex sheet discretizations [Kim et al. 2009; Pfaff et al. 2012; Brochu et al. 2012] are that we omit the Boussinesq approximation, and we use a low-resolution pressure gradient from the fluid simulation instead of the total acceleration of the surface. By tying the low-resolution simulation into our dynamics, our surface tracker is guaranteed to oscillate about a low-resolution surface representation. In contrast, a high-resolution vortex sheet discretization may easily drift from the simulation and become arbitrarily complicated. The derivation of our method also indicates that the vortex stretching and dilation terms of the vortex sheet equations are unrelated to the reduction of surface tracking errors; omitting these terms will only affect wave propagation speeds.

Because Equation (4.9) is based on the inadequacy of a low-resolution pressure gradient, the resulting dynamics are allowed to gracefully interact with an Eulerian fluid simulation without double-counting forces. As the resolution of the fluid simulation increases relative to that of the surface tracker, the effect of these additional fluid dynamics diminishes, until they disappear when the resolutions are equal. We found this approach to be an effective strategy for adding high-resolution dynamics to a low-resolution fluid simulation without requiring much computational overhead. Again our implementation uses an approximation to the Biot-Savart law to convert our angular acceleration into a velocity field, with details given in Section 4.5.

4.5 Implementation Details

In our fluid simulation implementation, we use a regular MAC grid fluid simulation with second order ghost fluid boundary conditions at the free surface [Bridson 2008]. We use the liquid-biased filter of Kim et al. [2009] to represent thin liquid features on the coarse fluid grid. We convert the rotational velocities in Section 4.4

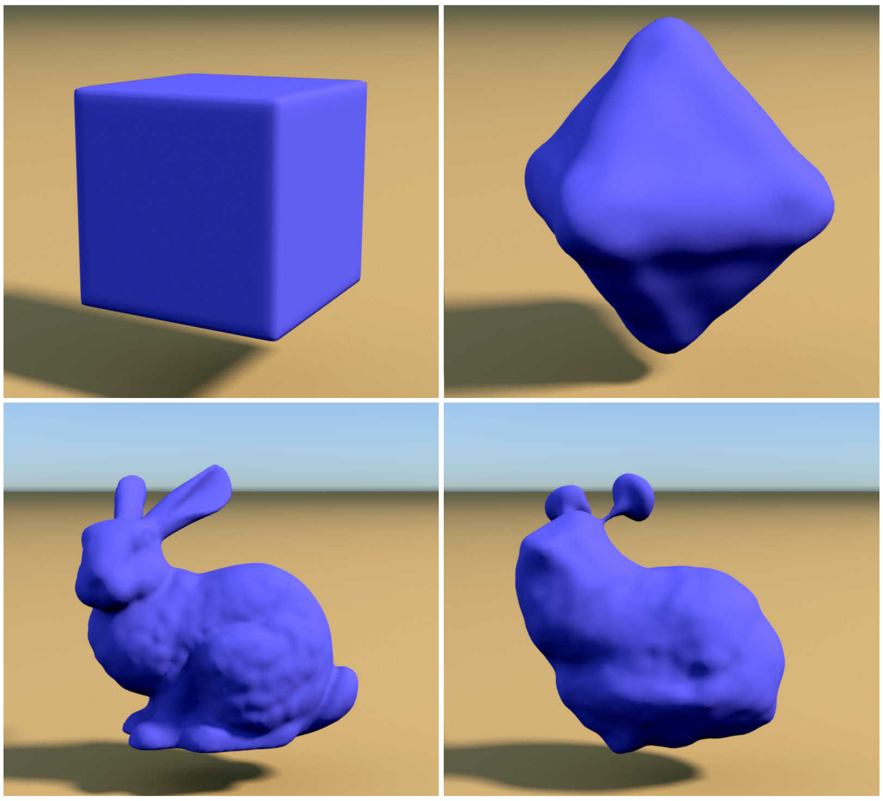


Figure 4.6: The technique in Section 4.4.2 can create detailed surface tension dynamics of a cube (top) and a bunny (bottom), even with a low simulation resolution.

into a velocity field using using the Biot-Savart Law with a finite kernel size, as in [Pfaff et al. 2012]. Because any size kernel will create a local rotation and will thus reduce the errors in the surface tracker, the kernel size is irrelevant when considering error minimization. Larger kernel sizes simply allow for lower-frequency surface rotations, which may have visual importance depending on the application. Both the first-order smoothing (Section 4.4.1) and second-order dynamics (Section 4.4.2) are integrated with a symplectic Euler method.

Once a pressure field has been computed in the fluid simulation, we compute its gradient and extrapolate both quantities past the free surface using the usual constant extrapolation in the normal direction. Subsequently, we use tri-linear interpolation when evaluating either quantity at points on the free surface. In some of our smoothing experiments, we found it useful to re-scale the pressure gradient by the difference between the extrapolated pressure evaluated at the interface and the outside air pressure instead of directly using the low-resolution pressure gradient. This adjustment has no effect in most cases near the fluid

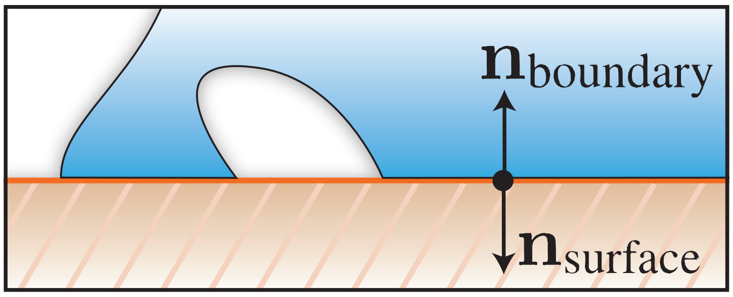


Figure 4.7: When the surface of a liquid (blue) contacts a solid obstacle (orange), the two normals should be anti-parallel.

surface, but it causes bubbles and air pockets deep below the liquid surface to smooth out more quickly.

4.5.1 Solid boundaries

Our high resolution surface tracker can cause kinks and bubbles along solid boundaries as well as along the free surface. When the surface pushes up against a solid obstacle, its surface normal should be anti-parallel that of the boundary (Figure 4.7). Using similar reasoning to Section 4.3, we arrive at the following update rule:

$$\mathbf{n}_{\text{surface}} \times \left(-\frac{\partial E_{\text{boundary}}}{\partial \mathbf{n}}\right) = \mathbf{n}_{\text{boundary}} \times \mathbf{n}_{\text{surface}} dA \tag{4.10}$$

We can also multiply the strength of this motion by the fluid pressure if we wish to create more drastic behaviors in high pressure regions. However, we declined this option in order to reduce sources of potential instability.

In practice, we switch from the free surface update rules in Section 4.4 to the solid boundary update rule for each point on the surface tracker that is within a given distance of the boundary. In this paper, we use a distance of three times the size of the minimum resolvable detail in the surface tracker.

We create a special case along corners where the low-resolution free surface borders a solid boundary, because we do not want a small change in position to result in a drastic change in motion. To remove the potential for such discontinuous gradients in the corners, we only allow rotations about an axis parallel to the boundary normal. We further add a small amount of Laplacian smoothing at the corners to compensate for errors that we ignore with this restriction.

We have tested our methods on both high resolution level set surface trackers as well as Lagrangian triangle mesh surface trackers. While the overall principles for integrating our error correction mechanism are independent of the tracker, we discuss some specific implementation details below.

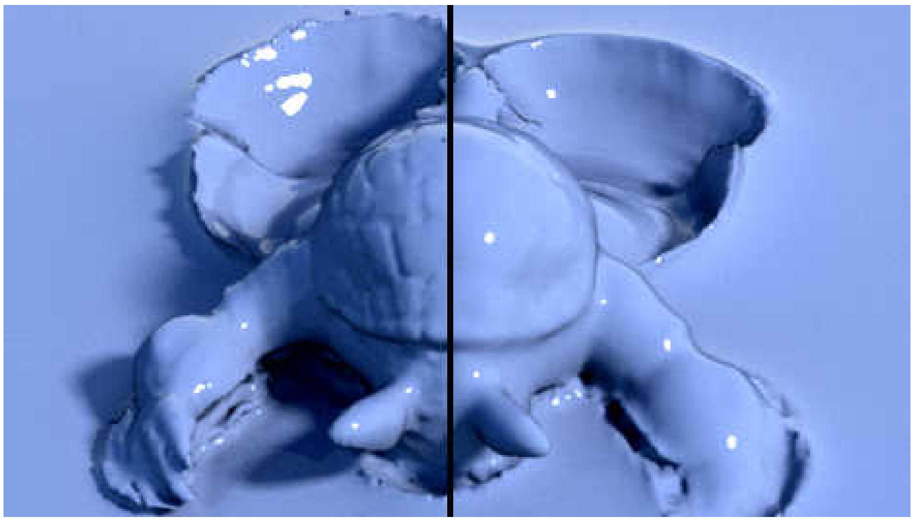


Figure 4.8: Our mesh-based smoothing method (left) preserves details in low-pressure regions, unlike mean curvature flow (right).

4.5.2 Level set surface tracker

We use the sparse level set implementation OpenVDB [Museth 2013] with an adaptive WENO5 scheme for level set advection, because it minimizes artificial diffusion artifacts. We used a narrow band the size of one fluid cell, and we evaluate the surface normal by taking the gradient of the signed distance function using central differencing. We compute free surface boundary conditions by down-sampling the level set function onto the fluid grid and then performing the ghost fluid method on the low-resolution distance function. We typically set the level set resolution four to eight times higher than that of the fluid simulation.

We compute our energy gradients (Equation (4.8) and Equation (4.9)) on a high-resolution sparse grid, and we use a smeared delta function with a radius of 1.5 level set cells to confine them to the surface, similar to Kim et al. [2009]. We then use a Biot-Savart kernel with a diameter of five level set grid cells to convert local rotations into detailed velocities, and we store the resulting velocities on a grid co-located with the level set samples. We up-sample the low-resolution fluid velocity field and add it to this high resolution velocity field in order to advect the level set. For computing dynamics, we also store, extrapolate, and advect the angular velocities on a grid co-located with the level set.

4.5.3 Mesh surface tracker

We use a Lagrangian triangle mesh surface tracker with a voxelization-based method for computing topology changes, as in Wojtan et al. [2010]. We create a low-resolution signed distance function from the triangle mesh as part of the topology-changing procedure, which we use for the ghost fluid method. We keep the topology change grid at the same resolution as the fluid simulation, and we maintain an average mesh resolution 4 to 5 times higher than that of the simulation grid.

We evaluate our energy gradients at the centroid of each triangle face using the geometric normal of each triangle. We follow Pfaff et al. [Pfaff et al. 2012] to compute a regularized Biot-Savart kernel with a diameter of four fluid simulation grid cells, and we store the resulting velocities on the mesh vertices. We also store and transport angular velocities by converting them to circulations on triangle faces, in the way of Stock et al. [2008]. We advect the mesh through the low-resolution velocity field of the fluid simulation with a fourth order Runge-Kutta method, and we advect the mesh through the high-resolution mesh velocity field using an Euler method.

4.5.4 Stability

We use symplectic Euler time integration for our surface fairing and dynamics algorithms, so we inherit the expected stability criteria. For instance, this means that our gravity waves should obey a standard CFL condition and that the surface tension time step should shrink with the spatial resolution to the power of 1.5. Although we are currently unable to provide a formula for the stability of our smoothing algorithm, we believe it has similar stability behavior to other volume-preserving fairing algorithms like bi-Laplacian smoothing.

Nevertheless, we noticed the level set implementation was remarkably stable in practice. In particular, we were able to take much larger time steps with our surface tension dynamics than with standard ghost fluid-based surface tension. Additionally, while excessively large correction forces caused high frequency oscillations, the method is able to recover quickly from short periods of instability.

Our mesh-based implementation is a bit more delicate to instabilities. While a grid-based level set has a guaranteed minimum distance between any two samples, centroids on a triangle mesh can be arbitrarily close together when folded into a thin sheet. The regularizer in the Biot-Savart kernel mentioned above helps to limit large velocities for close particles, and we also found it convenient to artificially limit the size of the error gradients by multiplying by a constant slightly less than one. For large α and β parameters, we used sub-cycling to integrate the surface tracker through several small time steps in between large fluid simulation time steps.

4.6 Results

We generated several didactic animations to illustrate the behavior of our method, which can be seen in the accompanying video. For illustrative purposes, these examples were computed with an exaggerated low resolution of 32^3 for the fluid simulation and 256^3 for the level set surface tracker. One obvious benefit of our method is that it can selectively remove high-resolution surface noise from a low-resolution simulation (Figure 4.5). The method also addresses the important problem of removing *topological* noise, such as high-resolution cavities and droplets. Figure 4.3 shows how a thin slot can persist in a fluid simulation as long as the gap is smaller than the width of a fluid cell. Simply smoothing the surface could eventually fix the problem, but not before first removing details from the entire simulation. In contrast, our method specifically targets the unphysical topological structure and rectifies it using either smoothing or splashes.

We found it difficult to directly compare our method with a Laplacian smoothing approach, because the methods behave quite differently and their parameters are not analogous. Qualitatively, we noticed that Laplacian smoothing erodes high curvature regions first and progressively smoothes away larger bumps over time, removing both surface artifacts and surface details without prejudice. Our smoothing method instead selectively erodes artifacts and ignores physically plausible surface features. In addition, our method has a convenient upper limit to the amount of surface smoothing: upon convergence it simply forces the surface tracker to exactly conform to the low-resolution fluid boundary conditions. The limit behavior of Laplacian smoothing, on the other hand, has nothing to do with natural fluid motion.

Figure 4.6 shows how our method can be used to supplement surface tension dynamics with a level set surface tracker. By computing surface tension waves on the surface tracker instead of the fluid grid, we were able to create surprisingly stable dynamics with good volume conservation properties.

Figure 4.8 shows how our surface fairing algorithm removes artifacts from a mesh-based surface tracker. The fluid simulation and all simple mesh operations (including our smoothing algorithm) were implemented in parallel, while our topology change code is a serial implementation. Consequently, the topology changes were the most computationally expensive part of this simulation, and the fluid pressure solve was the second most expensive. Our algorithm has negligible overhead for our mesh-based implementation. While our smoothing algorithm successfully removes artifacts from mesh-based simulations, the topology change algorithm of [Wojtan et al. 2010] conflicts with the subtle ripples generated by our wave dynamics algorithm. This is because it identifies high-resolution surface concavities as topological mistakes that must be re-sampled. In the future, we wish

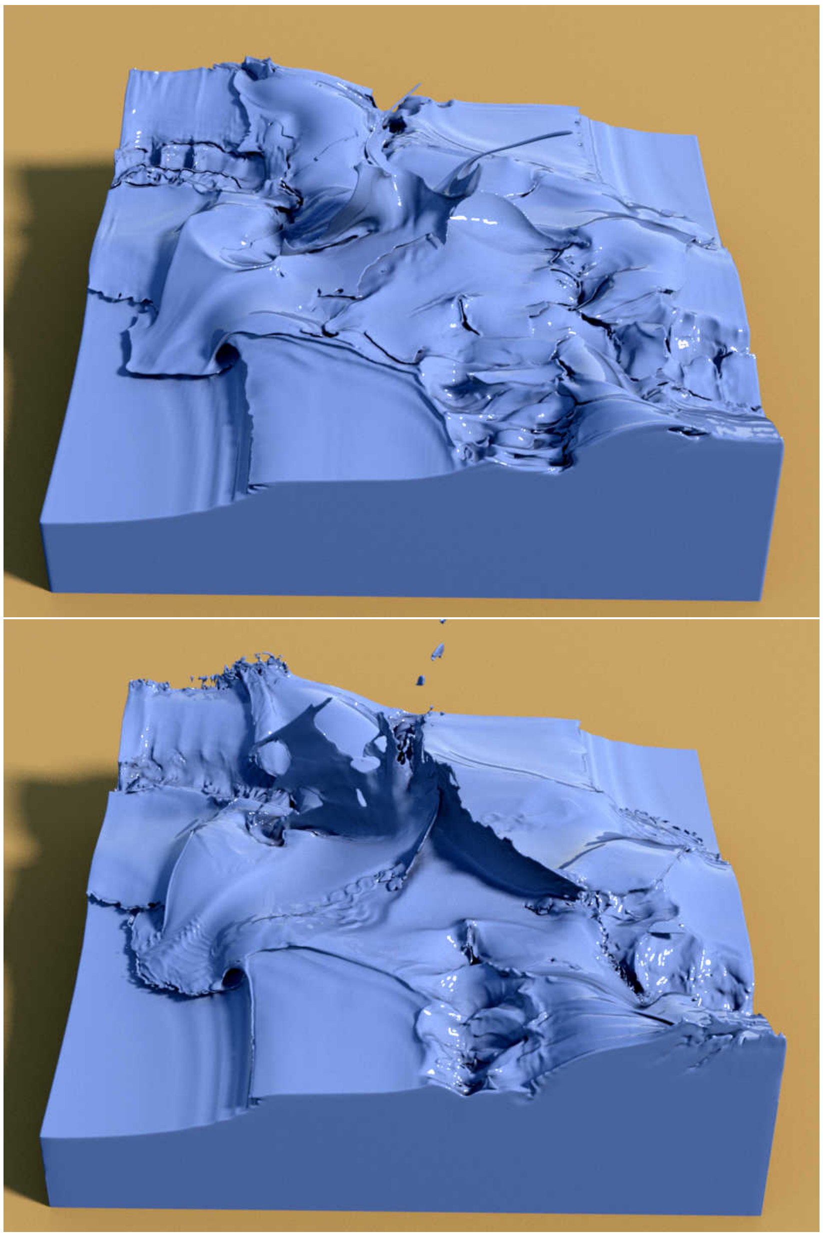


Figure 4.9: The original simulation (top) is enhanced by our wave dynamics (bottom). Notice how the detailed motion creates numerous topological artifacts holes and crevices in the right side of the original simulation while exhibiting unnatural jagged edges on the left. Our method cleanly removes these topological artifacts while significantly enhancing the liquid motions.

to modify this topology change algorithm or switch to a different one (e.g. [Brochu and Bridson 2009]).

Figure 4.1 shows a highly detailed object interacting with a fluid simulation. Using a high resolution level set without any smoothing successfully preserves the dragon's detailed surface features, but it also preserves many unphysical gashes and interpolation artifacts in the fluid surface. We found it impossible to remove these visual artifacts with Laplacian smoothing unless we also smoothed out the surface details of the dragon. In contrast, our method perfectly preserves the dragon's surface features while it is falling and only smooths them out when they splash around in areas of high pressure. By adding our error-reducing surface waves, we introduce highly detailed surface ripples and Kelvin-Helmholz instabilities while still avoiding unphysical artifacts. Similarly, the original simulation in Figure 4.9 exhibits several thin sheets and subsequent surface artifacts. As seen in our accompanying video, our method preserves details better than traditional smoothing while simultaneously removing unsightly errors and greatly enhancing the motion.

The most computationally expensive part of each of our level set simulations was the surface advection. The detail in our surface dynamics tends to create a more complicated velocity field, which slows down the conditionally-stable adaptive WENO advection. Biot-Savart calculation was not the bottleneck in our simulations, but it was a significant expense. This is because the velocity has to be evaluated at all cells in the high-resolution narrow band, instead of exactly on the surface. Our employment of relatively small kernel sizes disables lower frequency surface rotations, but it allows for reasonably efficient surface updates. These simulations were dominated by the surface tracker, and our method's complexity increases with surface tracker resolution. Consequently, the level set simulations augmented with our method took about 2.5 longer to simulate than an analogous high-resolution level set without any high-resolution dynamics (though sub-stepping makes our simulation proportionally slower). We believe this computational overhead is acceptable for our goals of removing surface artifacts while adding convincing dynamic details.

Detailed information about our simulations is listed in Table 4.1. Most of our simulations were run on a standard desktop computer with 8-cores and 64GB of RAM, while the Dragon (LS) and 4-way (LS) simulations were run on a 64-core server with 256GB of RAM.

4.7 Discussion

Our method solves the problem of removing high-frequency artifacts from a liquid surface tracker by identifying inconsistencies and explicitly removing them.

	Sim	Tracker	Base	Smth	Wave
Dragon (LS)	128^3	512^3	71	200	201
4-way (LS)	128^3	512^3	103	217	220
4-way (LS)	128^3	1024^3	594	—	*3260
4-way (Mesh)	96^3	480^3	106	106	—
Armadillo (Mesh)	96^3	360^3	74	85	—
Cube (LS)	64^3	64^3	50	—	6
Bunny (LS)	128^3	128^3	—	—	98

Table 4.1: Summary of timings (in minutes) for our error compensation algorithm. "Sim" and "Tracker" indicate the simulation and effective surface tracker resolution, respectively. "Base" indicates the high resolution surface tracker with no additional dynamics, "Smth" indicates our smoothing algorithm, and "Wave" corresponds to our wave dynamics. Dashes indicate simulations that were not run. A star indicates that four time steps were used per frame to ensure stability. Effective mesh resolution is based on the ratio between the average mesh edge length and the width of the computational domain. Timings exclude file I/O operations.

Minimizing all unphysical surface behaviors in one step will not only remove all visual artifacts, but it will also reduce the effective resolution of the surface tracker to exactly match that of the fluid simulation. By spreading out this minimization process over time, we introduce a physics-based smoothing process. Similarly, by using the errors as a potential energy, we gain surface wave behaviors that specifically remove artifacts. Our method aggressively removes high-frequency surface noise wherever the pressure gradient is large, and it only gradually removes them in regions where the internal fluid forces are smaller. Intuitively, this means that our method will preserve surface details and thin sheets while splashing around in the air, but it will quickly remove noise from standing water. Our wave dynamics will physically over-compensate for sudden changes in the pressure gradient during collisions, which tends to add even more detail to splashes and thin sheets.

The tuning parameter α in our surface fairing intuitively controls how much we will allow the surface to diverge from the low-resolution simulation. Large values ensure consistency between surface and simulation but prohibit details that fall below the simulation grid, such as thin sheets, from developing. Smoothing with too small of an α value may not remove artifacts quickly enough for a high-quality simulation. We set $\alpha = 20$ in all of our examples.

As mentioned in Section 4.4.2, the vortex sheet equations give us a useful guide for tuning the β parameter. We found that using this setting of $\beta = 2$ usually creates extremely subtle waves that only fill in the gaps of the low-resolution fluid simulation. On the other hand, we are free to artistically tune high-frequency details because our method is not restricted to faithfully reproduce any physical equation. We found that increasing β effectively boosts the sub-grid wave speed, leading to the creation of convincing (though technically unphysical) rolling motions as waves crash down. We used a value of $\beta = 2$ or $\beta = 6$ in all dynamics examples.

Our method is directly guided by the pressure gradient of a fluid simulation. On one hand, this constraint limits the guiding force field to a low resolution. On the other hand, simplified pressure gradients enhance the stability of rotational motion (see the Boussinesq approximation, for example), and our second-order dynamics allow for interesting turbulent motions even with a constant pressure gradient. In addition, allowing our method to be guided by a full fluid simulation causes many of the difficulties in handling solid boundaries to disappear.

We implemented our method in both level set and mesh surface tracking frameworks, and each of these discretizations has its own drawbacks. The level set method is quite robust, but it smooths out desirable details and currently requires a narrow band of at least one fluid cell in thickness to perform the liquid-biased filter. The mesh tracker uses less memory but is more delicate — samples can come very close together during fold-overs and thin sheet formation, re-sampling the surface is more involved (we use edge subdivisions and collapses as described in [Wojtan et al. 2011]), and robust and efficient handling of topological changes is still an open problem.

One inconsistency between our method and the underlying fluid simulation is that our local rotations enforce incompressibility for both air and water phases, while the fluid simulation ignores conservation of the air's volume. The large scale fluid simulation will freely allow large air pockets to collapse beneath the weight of the liquid, but our surface tracker dynamics preserve the volume of air bubbles smaller than a fluid cell. These air bubbles still experience buoyancy forces though, which cause them to rise up and break through the fluid surface. We found this behavior quite beautiful for the sub-grid fluid dynamics (Section 4.4.2), but the motion may be undesirable when performing surface smoothing (Section 4.4.1). We should be able to quickly detect and remove such small bubbles if the behavior is objectionable.

The memory complexity of standard grid-based fluid simulation techniques is proportional to the volume of the simulation, while our algorithm scales only with the surface area. As a result, we were able to achieve very high resolution simulations on a single computer without experiencing memory problems. With our dynamic wave method, we were also able to increase the apparent resolution of a simulation by factors of 4^3 or 8^3 while only requiring a constant factor more computation from the surface tracker. Nevertheless, we would like to further speed up these operations. The stability of the method may be improved by replacing the explicit Euler integration scheme with an implicit one. We may also be able to speed up or sidestep Biot-Savart summations by reformulating our energy gradient in terms of different control variables.

Additionally, we would like to investigate boundary handling in more detail. We are open to new strategies for handling fluid cells that exhibit both free-surface and solid boundary conditions, and we plan to experiment with higher resolution solid boundaries in the future.

Appendix: Energy gradient

We wish to compute the partial derivative of the following energy with respect to **n**:

$$E_{ST} = \int_{\partial\Omega} \mathbf{n}\cdot\nabla p \; dA - \sigma\mathbf{n}\cdot\nabla H \; dA$$

The first term is identical to Equation (4.1), and the gradient is computed analogously. The second term is a bit more complicated because H is a function of **n** — specifically, the mean curvature is equal to the divergence of the normal field. In the following derivation, $\langle\cdot,\cdot\rangle$ notation denotes the inner product, and we use the fact that $\langle f, \nabla\cdot g\rangle = -\langle\nabla f, g\rangle$ from integration by parts on a closed manifold domain:

$$\begin{aligned}
\frac{\partial}{\partial\mathbf{n}}\langle\mathbf{n},\nabla H\rangle\sigma dA &= \left(\langle\frac{\partial}{\partial\mathbf{n}}\mathbf{n},\nabla H\rangle + \langle\mathbf{n},\frac{\partial}{\partial\mathbf{n}}\nabla H\rangle\right)\sigma dA \\
&= \left(\nabla H + \langle\mathbf{n},\frac{\partial}{\partial\mathbf{n}}\nabla(\nabla\cdot\mathbf{n})\rangle\right)\sigma dA \\
&= \left(\nabla H - \langle\nabla\cdot\mathbf{n},\frac{\partial}{\partial\mathbf{n}}(\nabla\cdot\mathbf{n})\rangle\right)\sigma dA \\
&= \left(\nabla H + \langle\nabla(\nabla\cdot\mathbf{n}),\frac{\partial}{\partial\mathbf{n}}\mathbf{n}\rangle\right)\sigma dA \\
&= \left(\nabla H + \langle\nabla H,\frac{\partial}{\partial\mathbf{n}}\mathbf{n}\rangle\right)\sigma dA \\
&= (\nabla H + \nabla H)\sigma dA \\
&= 2\sigma\nabla H dA
\end{aligned}$$

Therefore:

$$\frac{\partial}{\partial\mathbf{n}}E_{ST} = (\nabla p - 2\sigma\nabla H)\,dA$$

Chapter 5

Generalized Non-reflecting Boundaries for Fluid Re-simulation

When aiming to seamlessly integrate a fluid simulation into a larger scenario (like an open ocean), careful attention must be paid to boundary conditions. In particular, one must implement special "non-reflecting" boundary conditions, which dissipate out-going waves as they exit the simulation. Unfortunately, the state of the art in non-reflecting boundary conditions (perfectly-matched layers, or PMLs) only permits trivially simple inflow/outflow conditions, so there is no reliable way to integrate a fluid simulation into a more complicated environment like a stormy ocean or a turbulent river.

This paper introduces the first method for combining non-reflecting boundary conditions based on PMLs with inflow/outflow boundary conditions that vary arbitrarily throughout space and time. Our algorithm is a generalization of state-of-the-art mean-flow boundary conditions in the computational fluid dynamics literature, and it allows for seamless integration of a fluid simulation into much more complicated environments. Our method also opens the door for previously-unseen post-process effects like retroactively changing the location of solid obstacles, and locally increasing the visual detail of a pre-existing simulation.

5.1 Introduction

Fluid simulations are indispensable for adding realism to large dynamic environments like open oceans. However, careful attention must be paid to boundary conditions if we wish to seamlessly integrate a fluid simulation into a larger scenario. Instead of making waves spuriously reflect off of invisible walls at the boundary of the simulation domain, we want outgoing waves to quietly dissipate as they exit the simulation; we refer to this desirable behavior as "non-reflecting" boundary conditions.

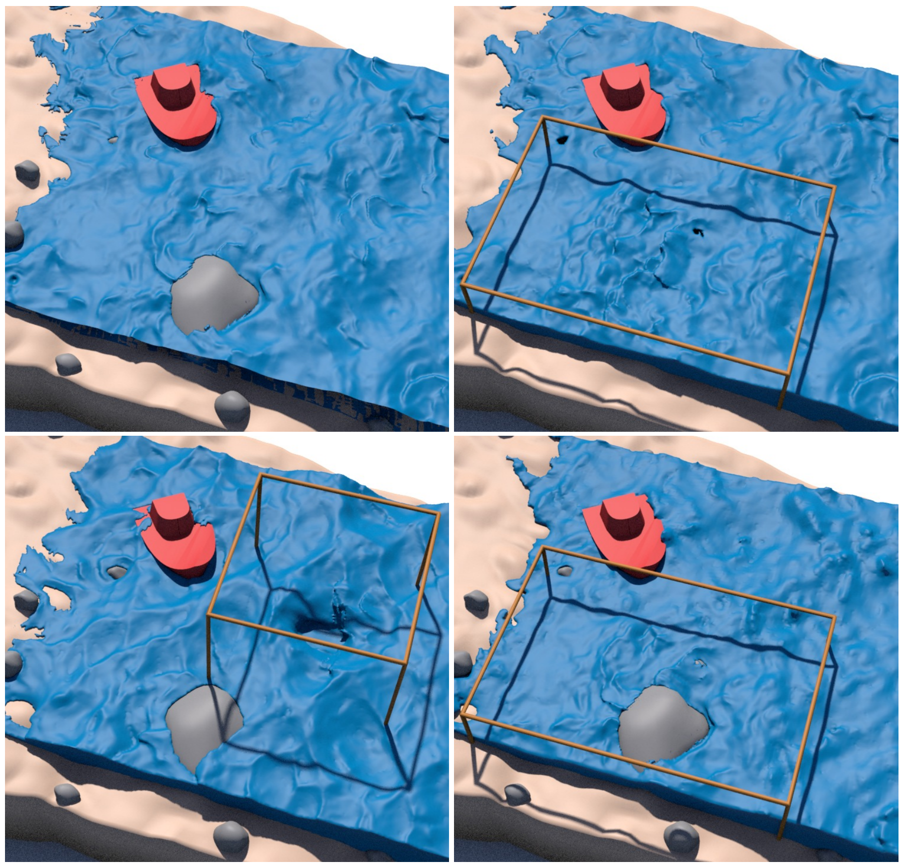


Figure 5.1: Given an input fluid simulation (top left), our algorithm can make local changes *retroactively* and seamlessly re-integrate them into the original fluid simulation. Here, we locally edit solid geometry (top right), add a cow splash (bottom left), or re-simulate a specific region at a higher resolution (bottom right). Please see our video.

The state of the art in non-reflecting boundary conditions is known as *perfectly-matched layers* or PMLs [Berenger 1994; Söderström et al. 2010]. At a high level, PMLs work by linearizing the equations of motion, performing a Fourier transform to identify outgoing waves, and exponentially damping the outgoing waves in a thin layer near the boundary of the simulation. Unfortunately, the state of the art in PMLs only permits trivially simple inflow/outflow conditions, like a stationary pool of water or a constantly translating stream (a steady-state mean flow). The state of the art leaves us with no reliable method for integrating a fluid simulation into a non-trivial (i.e., visually interesting) environment like a stormy ocean or a turbulent river.

In this paper, we generalize the state-of-the-art in non-reflecting boundary conditions and present several novel applications. Our contributions are as follows:

- The first PML method with spatially and temporally varying inflow/outflow boundary conditions
- The first derivation of the equations of motion of a perturbation relative to an existing fluid simulation
- The ability to add, remove, and adjust solid boundary obstacles retroactively in a fluid simulation
- The ability to locally re-simulate a fluid animation at a higher resolution (with more visual detail) as a post-process

5.2 Previous Work

The literature on fluid simulation for computer animation is vast. This discussion will focus primarily on our target problem of developing boundary conditions for fluid simulations, and on our target application of editing liquid simulations and efficiently re-simulating simulations.

Boundary conditions for fluid simulations. The problem of non-reflecting boundary conditions originated in computational physics. Berenger [1994] proposed the first method based on perfectly matched layers (PMLs) for the purpose of absorbing electromagnetic waves. PMLs were applied to computational fluid dynamics by Hu et al. [1996], starting with a linearized version of the Euler equations. Researchers subsequently applied PMLs to the non-linear Euler equations [Hu 2006] and Navier-Stokes equations [Hagstrom et al. 2005; Hu et al. 2008].

Hu [2001] and Bécache et al. [2003] showed that PMLs are only guaranteed to properly damp perturbations when the group and phase velocities are in consistent directions. While this disagreement between group and phase velocities never manifests for linear wave equations in a static reference frame, it becomes possible in the presence of a moving background flow, or with more complicated wave dispersion relationships. Their proposed solution to this problem is to apply a coordinate transformation that mathematically guarantees consistency of the group and phase velocities for all wave numbers.

Hu et al. [2008] showed how to create steady state mean flow PML boundary conditions for the Navier-Stokes equations. We generalize their work by allowing mean flows which can vary in time—an essential requirement for realistic computer graphics applications.

Within the field of computer graphics, several researchers proposed simple open boundary conditions for single-phase flows [Stam 1999; Fedkiw et al. 2001], but the wave reflection problem persisted for liquid simulations until the preliminary work of Söderström and Museth [2009] and their thorough follow-up work [Söderström et al. 2010] introduced PMLs for computer animation.

Nielsen and Bridson [2011] re-simulated the surface layer of a low resolution liquid at a higher resolution, using the low resolution simulation as a boundary condition. These guide shapes serve a similar purpose to our time-varying non-reflecting boundary conditions, especially when we inherit our boundary conditions from a lower-resolution simulation. However, their method is not based on PMLs and cannot prevent spurious boundary reflections.

Editing liquid simulations. Our non-reflecting boundary conditions allow a new method for locally editing a fluid animation. While editing a physics simulation is still a challenging problem, researchers in this area have developed several powerful editing techniques already.

Guiding methods [McNamara et al. 2004; Shi and Yu 2005; Thürey et al. 2006] allow more direct control than manipulating initial conditions, which may alleviate the need for going through a large number of iterations. Designing appropriate keyframes can be quite laborious, however, especially if the end result must look physically plausible.

Bhat et al. [2004] present a synthesis approach that allows editing of videos of flows exhibiting roughly stationary dynamics such as waterfalls and rivers. Pighin et al. [2004] go in a different direction and parameterize an Eulerian fluid simulation using advected radial basis functions. Simple edits may then be performed by manipulating flow streamlines. More recently, Pan et al. [2013] developed an interactive sketch-based approach to editing FLIP simulations. User edits are localized in space and time by encoding the edits as deformation fields that are then back-advected and applied with a smooth falloff. The final result is obtained by a guided offline simulation.

Raveendran et al. [2014] introduce a data-driven method for instantly generating new liquid simulations as an interpolation of the inputs using space-time blending. It does not allow arbitrary user edits, however.

Efficient re-simulation. Subspace methods provide a way of significantly reducing the degrees of freedom of the system by leveraging previous simulation data. Recently, Kim and Delaney [2013] addressed the inability of these methods to reproduce the input simulations with a cubature approach. This allowed parameters such as buoyancy, vorticity confinement, and timesteps to be varied and re-simulated efficiently. As with all model reduction methods, continuously changing boundary conditions such as moving solid obstacles or liquid surfaces remain a challenge.

To the best of our knowledge, Srinivasan and Malkawi [2007] provide the only existing method for automatic, localized fluid re-simulation. Their application is indoor airflow visualization for augmented reality. In a pre-computation step, airflow is simulated for a limited number of room topologies (placement and number of openings) using an Eulerian simulator. For each change in boundary condition (*e.g.* adding a window) the user extracts a small number of bounding boxes containing the most significantly changed grid nodes. These bounding boxes are the only areas re-simulated at runtime. Since bounding boxes are determined by running a full simulation, pre-computation time is quite significant, and also suffers from combinatorial explosion as the number of rooms is increased.

5.3 Perfectly Matched Layers

In this section, we will review the concept of perfectly matched layers (PMLs). We begin our derivation with the incompressible Navier-Stokes equations in conservation form:

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{0} \tag{5.1}$$

$$\mathbf{q} = \begin{pmatrix} 0 \\ \mathbf{u} \end{pmatrix} \tag{5.2}$$

$$\mathbf{F}(\mathbf{q}) = \begin{pmatrix} \mathbf{u} \\ \mathbf{u} \otimes \mathbf{u} + \frac{1}{\rho} p I \end{pmatrix} \tag{5.3}$$

Here, $\mathbf{u}$ denotes the usual three-dimensional velocity, ρ is density, p is pressure, and I is a 3×3 identity matrix. Notice that Equation (5.1) includes both the momentum equation and the zero-divergence constraint. Our exposition neglects viscosity and external forces for clarity, but they can easily be be included as in Söderström et al. [2010].

5.3.1 Basic PMLs

In the absence of any background flow, the perfectly matched layer will aim to exponentially damp $\mathbf{q}$ toward zero in a thin layer near the boundary. We apply the split variable approach [Berenger 1994] to split $\mathbf{q}$ into separate vectors associated with each spatial dimension $\mathbf{q} = \mathbf{q}_1 + \mathbf{q}_2 + \mathbf{q}_3$. The equation for the time evolution of $\mathbf{q}_1$ ($\mathbf{q}_2$ and $\mathbf{q}_3$ are analogous) is:

$$\frac{\partial \mathbf{q}_1}{\partial t} + \frac{\partial \mathbf{F}_1(\mathbf{q})}{\partial x} = \mathbf{0} \tag{5.4}$$

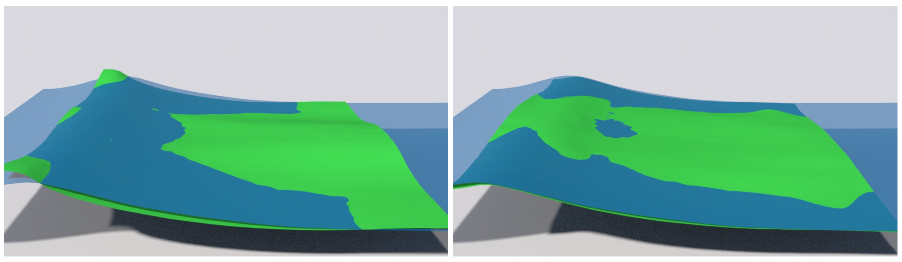


Figure 5.2: A simulation without (left) and with (right) perfectly matched layers (PMLs) with time-varying inflow/outflow boundary conditions. The simulation without PMLs exhibits interference patterns from wave reflections. The background flow is visualized in blue. For demonstration purposes this example does not include visual blending.

where

$$\mathbf{F}_1 = \begin{pmatrix} u_x \\ u_x^2 + p/\rho \\ u_x u_y \\ u_x u_z \end{pmatrix}, \mathbf{F}_2 = \begin{pmatrix} u_y \\ u_x u_y \\ u_y^2 + p/\rho \\ u_y u_z \end{pmatrix}, \mathbf{F}_3 = \begin{pmatrix} u_z \\ u_x u_z \\ u_y u_z \\ u_z^2 + p/\rho \end{pmatrix} \tag{5.5}$$

Our notation uses numerical subscripts to denote quantities and operators associated with split variables, while we use x, y, z subscripts to denote a velocity component in a particular spatial dimension. Note that we recover Equation (5.1) when we sum the split components defined by Equation (5.4).

We convert to the frequency domain by applying a Fourier transform $\partial/\partial t \rightarrow -i\omega$ and achieve a spatial stretching in the boundary layer (following Söderström et al. [2010]) with the transformation

$$\frac{\partial}{\partial x} \rightarrow \frac{1}{1 + i\sigma_1/\omega} \frac{\partial}{\partial x} \tag{5.6}$$

to get

$$-i\omega\tilde{\mathbf{q}}_1 + \frac{1}{1 + i\sigma_1/\omega} \frac{\partial \widetilde{\mathbf{F}_1(\mathbf{q})}}{\partial x} = \mathbf{0} \tag{5.7}$$

where the tilde notation indicates a quantity in the frequency domain. Here, $\sigma_1(x)$ is a spatially varying transfer function that depends only on x (not y or z). It will be used to damp waves traveling in the x-direction by setting it to zero in the simulation domain and ramping it up to a large positive value in the boundary layer. We then multiply through by $(1 + i\sigma_1)/\omega$ to get

$$-i\omega\tilde{\mathbf{q}}_1 + \sigma_1\tilde{\mathbf{q}}_1 + \frac{\partial \widetilde{\mathbf{F}_1(\mathbf{q})}}{\partial x} = \mathbf{0} \tag{5.8}$$

and finally transform back to the original domain with the inverse Fourier transform $-i\omega \rightarrow \partial/\partial t$

$$\frac{\partial \mathbf{q}_1}{\partial t} + \sigma_1\mathbf{q}_1 + \frac{\partial \mathbf{F}_1(\mathbf{q})}{\partial x} = \mathbf{0} \tag{5.9}$$

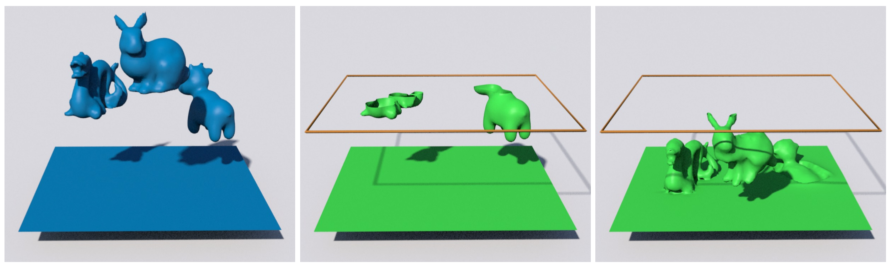


Figure 5.3: The background flow (left, blue) includes geometry outside of the simulated flow (middle, green). The upper boundary of the new simulation is indicated by the wire rectangle on the right. Nevertheless, new geometry (the bunny) flows in through the upper boundary as the simulation progresses.

Numerically integrating this equation will exponentially damp $\mathbf{q}_1$ towards zero in the boundary layer. Summing up each of the split components at the end of each time step will recover $\mathbf{q}$.

5.3.2 Background Flows

By damping to $\mathbf{q} = \mathbf{0}$ near the boundary, Equation (5.9) implies that the simulation is located in the middle of a perfectly static fluid. To allow for more interesting background motions, Hu et al. [2008] damp towards a background flow $\overline{\mathbf{q}}$ instead of towards zero. They interpret $\mathbf{q}$ as the summation of the background flow $\overline{\mathbf{q}}$ and a perturbation flow $\mathbf{q}'$:

$$\mathbf{q} = \overline{\mathbf{q}} + \mathbf{q}' \tag{5.10}$$

where $\mathbf{q}$ and $\overline{\mathbf{q}}$ are solutions to Equation (5.1). Note, however, that because the Navier-Stokes equations are non-linear, $\mathbf{q}'$ will generally *not* be a solution to Equation (5.1), making the problem of solving for the motion of $\mathbf{q}'$ substantially more complicated.

Hu [2006] simplifies this problem by assuming that the background flow is already in steady state, i.e., $\partial\overline{\mathbf{q}}/\partial t = \mathbf{0}$, which implies $\partial\mathbf{q}'/\partial t = \partial\mathbf{q}/\partial t$ by Equation (5.10). Thus, under the steady-state assumption, the dynamics of $\mathbf{q}'$ are just $\partial\mathbf{q}'/\partial t + \nabla\cdot\mathbf{F}(\mathbf{q}) = \mathbf{0}$. Applying the PML transformations above gives us

$$\frac{\partial \mathbf{q}_1'}{\partial t} + \sigma_1 \mathbf{q}_1' + \frac{\partial \mathbf{F}_1(\mathbf{q})}{\partial x} = \mathbf{0} \tag{5.11}$$

which is essentially the same as Equation (5.9), except it damps $\mathbf{q}'$ to zero near the boundaries instead of damping the entire $\mathbf{q}$. Intuitively, this drives $\mathbf{q}$ toward the background flow $\overline{\mathbf{q}}$ near the boundary of the simulation domain. This represents the state of the art in non-reflecting boundary conditions, which we will improve upon in the next section.

5.4 Time-varying Background Flows

While steady-state background flows are convenient for applications like the analysis of airfoils, they are extremely limiting for computer graphics applications. The steady-state assumption is simply insufficient if we wish to have our simulation live within a more natural environment, like an undulating ocean or a turbulent river. In this section we explain our main contribution of achieving time-varying background flows with perfectly matched layers.

We again divide our main simulation variable $\mathbf{q}$ into a background flow $\overline{\mathbf{q}}$ and a perturbation $\mathbf{q}'$, as in Equation (5.10), but we remove the assumption that $\partial\overline{\mathbf{q}}/\partial t = \mathbf{0}$. As a consequence, we can no longer equate $\partial\mathbf{q}'/\partial t$ to $\partial\mathbf{q}/\partial t$, and we cannot make use of the previous derivation.

However, since both $\mathbf{q}$ and $\overline{\mathbf{q}}$ are solutions to the Navier-Stokes equations, we can write

$$\left[\frac{\partial\mathbf{q}}{\partial t} + \nabla\cdot\mathbf{F}(\mathbf{q})\right] - \left[\frac{\partial\overline{\mathbf{q}}}{\partial t} + \nabla\cdot\mathbf{F}(\overline{\mathbf{q}})\right] = \mathbf{0} \tag{5.12}$$

We then regroup the terms

$$\left[\frac{\partial\mathbf{q}}{\partial t} - \frac{\partial\overline{\mathbf{q}}}{\partial t}\right] + \left[\nabla\cdot\mathbf{F}(\mathbf{q}) - \nabla\cdot\mathbf{F}(\overline{\mathbf{q}})\right] = \mathbf{0} \tag{5.13}$$

and redistribute the linear derivatives.

$$\frac{\partial\left[\mathbf{q}-\overline{\mathbf{q}}\right]}{\partial t} + \nabla\cdot\left[\mathbf{F}(\mathbf{q}) - \mathbf{F}(\overline{\mathbf{q}})\right] = \mathbf{0} \tag{5.14}$$

We next substitute the definition of $\mathbf{q}'$ to derive the dynamics of the perturbation.

$$\frac{\partial\mathbf{q}'}{\partial t} + \nabla\cdot\left[\mathbf{F}(\mathbf{q}) - \mathbf{F}(\overline{\mathbf{q}})\right] = \mathbf{0} \tag{5.15}$$

Note that, although the resulting equation is simple, we made no assumptions in its derivation. In particular, we made no linearizations, and $\mathbf{q}'$ is not restricted to be a solution to the Navier-Stokes equations. We then apply the same splitting and coordinate transformations above to recover the PML-specific dynamics.

$$\frac{\partial\mathbf{q}_1'}{\partial t} + \sigma_1\mathbf{q}_1' + \frac{\partial\left[\mathbf{F}_1(\mathbf{q}) - \mathbf{F}_1(\overline{\mathbf{q}})\right]}{\partial x} = \mathbf{0} \tag{5.16}$$

Integrating this equation will again damp $\mathbf{q}_1'$ toward zero near the boundaries, but it will also allow $\overline{\mathbf{q}}$ to vary arbitrarily over space and time. Intuitively, $\mathbf{q}$ will continually be damped towards $\overline{\mathbf{q}}$, but now $\overline{\mathbf{q}}$ is a moving target that is free to erratically splash up and down or flow in and out of the domain.

Our derivation of Equation (5.16) is similar to the one of Hu [2006]. Hu goes on to assume a pseudo mean flow that satisfies steady-state Navier-Stokes, however, our application crucially depends on having a time-varying background flow, which may not be as obviously useful for CFD applications.

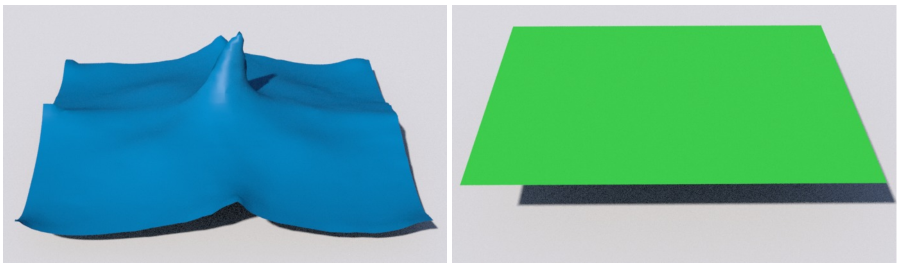


Figure 5.4: Even when the background flow (left, blue) is very lively, our algorithm is able to produce a perfectly static pool (right, green).

5.4.1 An Addendum for Liquid Surfaces

Equation (5.16) naturally damps differences in the velocity field near the boundary, but it will not remove differences in the liquid surface geometry. In order to drive the *entire physical state* toward the time-varying background flow, we may also add a small damping term to the free surface geometry. In the case of a level set [Osher and Fedkiw 2006], which is what we used in all of our experiments, this can be expressed

$$\frac{\mathrm{D}\phi}{\mathrm{D}t} + \gamma\left(\phi - \overline{\phi}\right) = 0 \tag{5.17}$$

where γ is a transfer function similar to σ that ramps upward in the boundary layer, ϕ is the level set function representing the geometry of the liquid surface, $\overline{\phi}$ is the level set of the background flow and $\mathrm{D}/\mathrm{D}t$ is the material derivative. In our examples, we found that the free-surface advection combined with the copying operation described below was accurate enough to safely set $\gamma = 0$, but we document this concept for completeness. We did not experiment with other surface trackers such as triangle meshes or particles.

This damping operation removes continuous deformations of the liquid surface geometry, but it cannot change topology, like when an inflowing velocity field brings entirely new surface geometry through the boundary. In such cases, we must explicitly add the new geometric components to our surface tracker. We do this by copying all liquid geometry from the background flow within a layer with width dependent on a CFL condition (usually 3 cells) beyond the simulation boundary, and we extrapolate the velocity from these regions as well. This way, the liquid surface advection algorithm naturally carries new geometry components in through the boundaries of the simulation. See Figure 5.3 for a proof of concept.

5.5 Implementation Details

We implemented our method as a set of plug-ins for Houdini [Side Effects Software 2016]. The source code for these plug-ins as well as an example Houdini scene file

Algorithm 5.1 Pseudocode for our one time step of our algorithm.

```
1: while sub-cycling do
2:     Compute sub-cycle time step size $\Delta t_{\text{sub}}$
3:     Advect $\mathbf{q}'$ with step size $\Delta t_{\text{sub}}$
4:     Damp $\mathbf{q}'$ with step size $\Delta t_{\text{sub}}$ (Equation (5.18))
5: end while
6: Add Body Force
7: Pressure Projection
8: Extrapolate $\mathbf{q}$
9: Advect Level Set
```

is included in the supplementary material.

Our implementation updates $\mathbf{q}'$ by time-splitting. We analytically integrate the middle (damping) term of Equation (5.16):

$$\mathbf{q}' \leftarrow \mathbf{q}' e^{-\sigma_1 \Delta t} \tag{5.18}$$

The rightmost term of Equation (5.16) encodes advection, the pressure projection, and the pressure update. Like most solvers in computer graphics, we apply time splitting and numerically integrate each term separately. Whenever our solver needs to evaluate an element of $\mathbf{q}$, we sum up each of the split components with $\mathbf{q} = (\mathbf{q}_1' + \mathbf{q}_2' + \mathbf{q}_3') + (\overline{\mathbf{q}}_1 + \overline{\mathbf{q}}_2 + \overline{\mathbf{q}}_3)$.

The order of our time splitting almost exactly follows Söderström et al. [2010], including the advection, the pressure projection, and the addition of conservative body forces like gravity. Because the explicit advection algorithm proposed by Söderström et al. [2010] is conditionally stable, we perform multiple sub-cycles of the advection and damping routines with the maximum stable time step based on the CFL condition [Bridson 2008]. We track the liquid free-surface using Houdini's particle level set method [Osher and Fedkiw 2006]. Pseudocode for one simulation time step can be seen in Algorithm 5.1.

To achieve good damping performance for the PMLs, it is important to pick good parameters. Söderström et al. [2010] performed a rather thorough experimental study of the effect of different choices for $\sigma(x)$, $\sigma_{\max}$ and PML width. Although our setting is slightly different from theirs, we found that basing our parameters on their findings worked well in practice.

In our examples, we set each PML's width to 8% of the simulation domain, and we use the transfer function $\sigma(x) = \sigma_{\max}(x/L)^3$, where x is the distance from the end of the simulation domain, L is the width of the PML and $\sigma_{\max} = 77$. We set the geometry blending coefficient γ to 0.

The PML's exponential damping in the boundary layer is essential for efficiently removing perturbations, but the change is a bit too sudden for the purpose of compositing a new simulation into an existing one as in Figure 5.1. For this visual

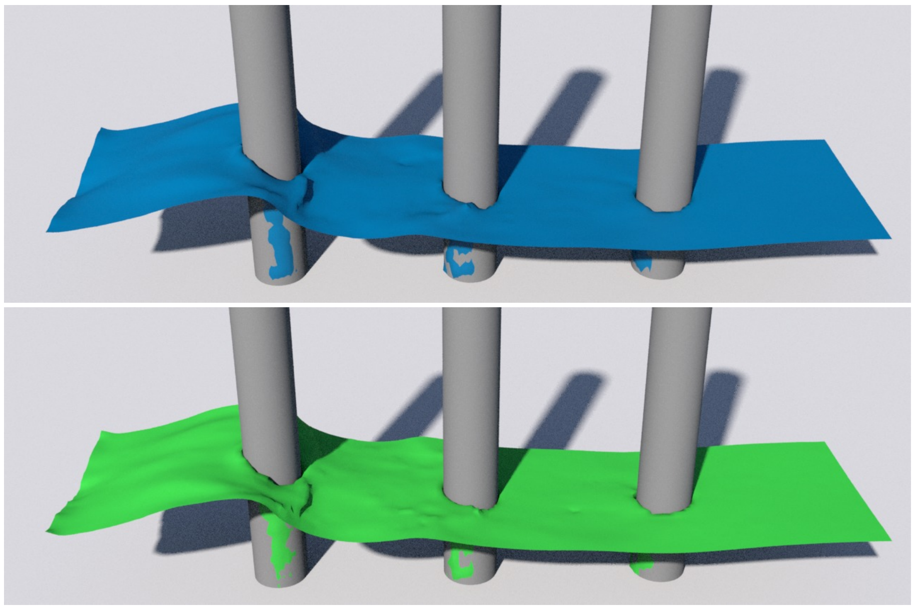


Figure 5.5: Our simulation (bottom, green) successfully reproduces the background flow (top, blue) when there are no perturbations.

transition, we remove the PML geometry and linearly blend the surface geometry in the outer $6-12\%$ of the simulation domain as a post-process.

During each time step, we fetch part of the background flow $\overline{\mathbf{q}}$ from disk. To make the computational complexity independent of the size of the background simulation, we modified OpenVDB [Museth 2013] to support sparse *out-of-core* grids.

5.6 Evaluation

We made a small collection of examples to show the robustness of our algorithm. We first verify that our solver can compute the correct behavior even when the correct motion $\mathbf{q}$ deviates far from the background flow $\overline{\mathbf{q}}$. Figure 5.4 shows that we can recreate a completely static pool even if the background flow is very lively. We do this by effectively deleting a falling sphere from the background flow, and simulating the result as if the original splash never took place ($\mathbf{q}' = -\overline{\mathbf{q}}$). Note how our resulting motion is independent of the background flow, which would not have been possible if we assumed the perturbation $\mathbf{q}'$ was small or if we linearized the Navier-Stokes equations about $\overline{\mathbf{q}}$.

In the other extreme, if we do not make any perturbations at all ($\mathbf{q}' = \mathbf{0}$), then our method reproduces the background flow $\overline{\mathbf{q}}$. Figure 5.5 shows that our method reproduces the expected motion in the absence of perturbations, and it does not drift away from the original simulation.

Figure 5.3 shows how our method advects new surface geometry into the simulation when dictated by the background flow. This behavior is important for merging simulations together, like when droplets spray into the simulation domain from somewhere outside.

5.7 Applications

Our method can create a simulation that appears to be part of a much larger fluid domain. Figure 5.2 (right) shows how we can use boundary conditions from a gently sloshing pool to create a new simulation set in the middle of a larger simulation. We first notice that ocean waves roll in and out through the domain boundary; this behavior is impossible with previous approaches. Next, we see that even though our simulation creates a large splash, the waves are absorbed by the boundaries. If we do not use our PMLs Figure 5.2 (left), we see obvious wave reflection artifacts.

Next, we can retroactively change parts of an existing simulation without rerunning the entire simulation. We begin with a complicated beach flow that was computed previously. In the top right of Figure 5.1, we create a small simulation domain around a solid obstacle that we wish to change, using the pre-computed simulation in the top left of Figure 5.1 as the background flow $\overline{\mathbf{q}}$. We completely remove the obstacle, locally creating a new flow. The combination of non-reflecting boundaries and our novel time-varying background flow allow us to seamlessly merge this simulation together with the larger one. We also perform a similar process in the bottom left of Figure 5.1, by re-simulating a new splash into the beach simulation as a post-process.

Our method can also retroactively increase the resolution of a portion of a simulation, allowing users to “zoom in” as a post-process. In the bottom right of Figure 5.1, we create a higher resolution simulation domain where we wish to add more detail to the beach simulation. We then use a lower-resolution flow as the $\overline{\mathbf{q}}$ in our boundary conditions, and we run the new simulation in a small subset of this domain at a much higher resolution. Again, our method allows the two simulations to be seamlessly blended together at the simulation boundaries.

Although we view our method as a working prototype and have not optimized it for performance, we list performance figures in Table 5.1. We believe that the low-resolution examples simulate $\overline{\mathbf{q}}$ significantly faster than $\mathbf{q}'$ because $\mathbf{q}'$ has substantial I/O overhead when reading the boundary conditions from the pre-computed $\overline{\mathbf{q}}$. As we localize the flows at higher resolutions, however, the $\mathbf{q}'$ simulations are much faster than the original $\overline{\mathbf{q}}$, allowing many post-processing passes once an initial simulation is computed.

5.8 Discussion

This work represents a significant generalization of the state of the art in non-reflecting boundary conditions for fluid simulation. Our method opens the door for novel post-process simulation editing and enhancement, and we presented prototypes for retroactively editing solid geometry and increasing simulation resolution.

The theory behind perfectly matched layers guarantees exponential damping of waves in the boundary layer for linear partial differential equations. However, not much is known about guarantees (if any exist) for non-linear equations like ours. The method seems to work exceptionally well in practice, but further theoretical development on this topic should be conducted. We observe that PMLs for the Navier-Stokes equations are dramatically more efficient than the naïve approach of damping the velocity field near the boundary, because they require a far smaller damping region and thus less memory and overall computation time.

Hu [2001] and Bécache et al. [2003] showed that theoretical guarantees on PMLs are only valid when the flow's group and phase velocities are in the same direction. We observed this problem for the scalar wave equation but never for Navier-Stokes. We suspect this is due to the natural frequency dispersion of surface water waves and numerical diffusion in our Navier-Stokes solver. As a result of our observations, we found it unnecessary to implement any of the corrections in the literature for 3D free-surface flows.

In our implementation, highly complex solid obstacles that intersected the boundary layer would occasionally prevent perturbations from quickly damping out. Less complex solid obstacle geometry, however, posed no difficulties (as exhibited by Figure 5.1). We have yet to conclude whether this is a general theoretical problem or one specific to our implementation.

Our current implementation assumes that the background flow $\overline{\mathbf{q}}$ is a solution to the Navier-Stokes equations discretized by our split-variable solver. We would like to remove this restriction in the future by allowing $\overline{\mathbf{q}}$ to be an arbitrary vector field (*i.e.* from a different fluid solver, or even from an unphysical artist-designed flow). We believe that this can be made possible by adding source terms to the equation of motion and applying the appropriate PML transforms.

The fact that advection is split over three equations prevents an easy extension to semi-Lagragian methods and also to FLIP. This is true even for a non-split variable approach as presented in Hu [2006]. We see it as a very fruitful avenue of future work to investigate how our method could be reconciled with such methods.

A property of our method is that it tends to preserve artifacts present in the input unless they are perturbed. Figure 5.5 exhibits small bumps in the input animation due to our use of a particle level set. Similarly, the input simulation

Example	Resolution	Run Time
Figure 5.3 $\overline{\mathbf{q}}$	$100 \times 100 \times 100$	25m 33s
Figure 5.3 $\mathbf{q}'$	$100 \times 50 \times 100$	45m 03s
Figure 5.4 $\overline{\mathbf{q}}$	$133 \times 53 \times 33$	10m 28s
Figure 5.4 $\mathbf{q}'$	$133 \times 53 \times 33$	21m 43s
Figure 5.5 $\overline{\mathbf{q}}$	$50 \times 50 \times 50$	6m 32s
Figure 5.5 $\mathbf{q}'$	$50 \times 50 \times 50$	11m 23s
Figure 5.2 (left) without PML	$53 \times 32 \times 60$	4m
Figure 5.2 (right) with PML	$53 \times 32 \times 60$	4m
Figure 5.1 (top left) $\overline{\mathbf{q}}$	$453 \times 100 \times 307$	15h
Figure 5.1 (top right) $\mathbf{q}'$	$227 \times 67 \times 153$	3h 47m
Figure 5.1 (bottom left) $\mathbf{q}'$	$133 \times 107 \times 133$	47m 17s
Figure 5.1 (bottom right) $\mathbf{q}'$	$227 \times 67 \times 153$	3h 30m

Table 5.1: Performance statistics for our absorbing boundaries. The background flow and perturbation flow (the new simulation) are indicated by $\overline{\mathbf{q}}$ and $\mathbf{q}'$, respectively.

in Figure 5.1 contains subtle "tendril"-like artifacts, which we suspect are caused by the wave generation algorithm linearly blending the fluid velocity field with a procedural vector field.

We imagine several extensions to our methods for retroactively improving a simulation. For example, we hope to combine our approach with an adaptively re-sizing simulation domain, allowing us to locally halt the simulation where the perturbation flow has damped out, and to adaptively expand the simulation domain where interesting new flows persist.

Chapter 6

Conclusion

In this thesis we have presented three novel contributions (as described in Chapters 3 to 5) to the field of liquid simulation for computer graphics. In this section we summarize these contributions and conclude the thesis by listing work relevant to ours that was published concurrently or after the work in this thesis

In Chapter 3 we presented a novel approach that takes a sequence of arbitrary closed surfaces and produces as output a temporally coherent sequence of meshes augmented with vertex correspondences. The output of our algorithm is useful for a variety of applications such as (dynamic) displacement maps, texture propagation, template-free tracking and morphs. We have also demonstrated the robustness of the method to parameters as well as input. In the future we would like to extend the method to handle non-closed surfaces, as well as explore problem-specific applications (*e.g.* tracking of biological cell data) of our general-purpose framework.

In Chapter 4 we presented a new approach to combating the artifacts that result from using mismatched degrees of freedom between a coarse volumetric liquid simulation and a high-resolution surface tracker. Unlike previous approaches, we precisely characterize the artifacts with a surface-based energy that is minimized when the surface-tracker is in physically valid state. This happens precisely when the pressure gradient and the surface normal are anti-parallel. Reducing this energy using gradient descent, we derive both smoothing and dynamics that quickly remove the disturbing surface artifacts or turn them into surface ripples, respectively. We also show a striking similarity between our dynamics, which are derived by interpreting our energy as a physical potential, and the vortex sheet equations.

In Chapter 5 we develop a novel method that enables a new workflow for liquid simulation artist. Instead of having to re-simulate everything anew every time a change is desired, they may instead perform local, incremental edits of an existing liquid simulation without worrying about wave reflections or messing up parts of the simulation that they are already happy with. To facilitate this new workflow, we derive the equation of motion for the perturbation of an existing simulation and extend existing perfectly matched layer theory to damp this perturbation to zero at the seam between the new and existing simulations. This enables us to remove solid obstacles, add new splashes and locally increase the resolution at a fraction of the cost of re-simulating the whole simulation anew.

6.1 Recent and concurrent work

Liquid simulation is a very active area of research in computer graphics, and it should come as no surprise that several works related to the research presented in this thesis were developed either concurrently to or after we initially published our results. In this section, we mention some of the most important results that a reader of this thesis should be aware of.

Stam and Schmidt [2011] showed how to derive the normal velocity of an implicit surface, however, to pin down the tangential velocity they had to make an assumption. To facilitate translational motion, Stam and Schmidt [2011] chose the assumption that the normal at any point should be invariant in time. After we published our mesh-tracking paper [Bojsen-Hansen et al. 2012], Fujisawa et al. [2013] showed that this assumption makes the original algorithm sensitive to rotations. They instead substitute an assumption of time-invariant *curvature*, which improves results with rotations. More recently, Gagnon et al. [2016] presented a completely different approach to texturing liquids using dynamically re-sampled tracker particles with local coordinate systems and overlapping textures.

The year after we published our error compensation algorithm [Bojsen-Hansen and Wojtan 2013], Edwards and Bridson [2014] presented a liquid simulation method using a finite element discretization and very high-order polynomial basis functions to practically (though, strictly speaking not theoretically) resolve all the details an embedded triangle mesh surface tracker. Goldade et al. [2016] published a direct follow-up to our work. They derived a more efficient version of our smoothing algorithm by blending between the high resolution surface and a low-pass filtered version instead of solving a partial differential equation, as we do. This year, Da et al. [2016] published a very interesting result that shows, under certain assumptions, that it is possible to completely do away with an underlying volumetric discretization and simulate three-dimensional fluids using *only* a surface triangle mesh.

Appendix A

Fluid simulation

In this appendix we will briefly sketch the steps involved in simulating a incompressible fluid using the incompressible Navier-Stokes equations (*cf.* Equations (2.28) and (2.30)). We refer to Bridson [2008] for a more complete exposition.

The first step is to discretize the time derivative $\partial \mathbf{u}/\partial t$ in the momentum equation using forward Euler as described in Section 2.3.4.

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t\Big(-(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f}\Big) \tag{A.1}$$

We have intentionally avoided discretizing the other terms (inside the parentheses) for now. Equation (A.1) is a bit of a mouthful to handle in one go, so following standard practice we are going to split each time step into a series of intermediate steps (corresponding to each term of the momentum equation) that add up to the full time step. Such methods are called *fractional-step* or *time-splitting* methods. Unlike the continuous setting where everything is "solved simultaneously", we are now have to choose in which order to solve the four terms inside the parentheses in Equation (A.1). Regardless of the order we choose, we will be incurring an error on the order of $O(\Delta t)$, so a priori any order will do. However, some errors are less visually disturbing than others. Below we have chosen an order that gives a good balance between efficiency and acceptable error. An important feature of the order that we choose is that we always finish the time step by enforcing the incompressibility constraint such that $\nabla \cdot \mathbf{u}^n = 0$ at the beginning of every time step.

In the first step we apply viscosity to the divergence-free field $\mathbf{u}^n$

$$\mathbf{u}^V \leftarrow \mathbf{u}^n + \Delta t\, \nu\nabla^2\mathbf{u^n} \tag{A.2}$$

This can either be solved with explicit time integration as above or implicitly. Either way, it works out very similar to the Poisson problem for pressure below (see Bridson [2008] for more details). Next, we advect the post-viscosity field $\mathbf{u}^V$ in the

Algorithm A.1 Typical simulation loop of incompressible Navier-Stokes.

```
1: for frame $n = 1 \to N$ do
2:     Apply viscosity to $\mathbf{u}^n$ to obtain $\mathbf{u}^V$
3:     Advect $\mathbf{u}^V$ in $\mathbf{u}^n$ to obtain $\mathbf{u}^A$
4:     Add body force accelerations to $\mathbf{u}^A$ to obtain $\mathbf{u}^B$
5:     Solve the Poisson problem $-\Delta t \nabla \cdot \frac{1}{\rho} \nabla p = -\nabla \cdot \mathbf{u}^B$
6:     Update $\mathbf{u}^B$ with the negative pressure gradient to obtain $\mathbf{u}^{n+1}$
7:     Advect the liquid surface in $\mathbf{u}^{n+1}$
8: end for
```

divergence-free field $\mathbf{u}^n$

$$\mathbf{u}^A \leftarrow \mathbf{u}^V - \Delta t \left(\mathbf{u}^{\mathrm{n}} \cdot \nabla\right) \mathbf{u}^V \tag{A.3}$$

This is typically solved using methods based on the method of characteristics such as semi-Lagrangian advection [Stam 1999] or FLIP [Zhu and Bridson 2005]. Next, we add accelerations due to body forces $\mathbf{f}$ such as gravity to the post-advection field

$$\mathbf{u}^B \leftarrow \mathbf{u}^A + \Delta t \mathbf{f}. \tag{A.4}$$

Finally, we need to update the velocity with the pressure gradient to enforce the incompressibility constraint $\nabla \cdot \mathbf{u}^{n+1} = 0$ at the end of the time step

$$\mathbf{u}^{n+1} \leftarrow \mathbf{u}^B - \Delta t \frac{1}{\rho} \nabla p \tag{A.5}$$

According to Equation (2.32) the pressure that enforces the incompressibility constraint can be found by solving the Poisson problem

$$-\Delta t \nabla \cdot \frac{1}{\rho} \nabla p = -\nabla \cdot \mathbf{u}^B \tag{A.6}$$

This Poisson equation is typically solved as in Section 2.3.4. The final step is to move the position of the liquid interface (assuming we are simulating a liquid) according to the new velocities $\mathbf{u}^{n+1}$. This is the topic of Section 2.10. The whole simulation loop is summarized in Algorithm A.1.

Bibliography

ANDO, R., THUEREY, N., and WOJTAN, C. 2015. A Stream Function Solver for Liquid Simulations. In *ACM Transactions on Graphics (SIGGRAPH)* 34.4, 53:1–53:9. [1, 22].

ANGELIDIS, A. and NEYRET, F. 2005. Simulation of Smoke Based on Vortex Filament Primitives. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. ACM, 87–96. [26].

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., and STRAIN, J. A. 2006. A semi-Lagrangian contouring method for fluid simulation. In *ACM Transactions on Graphics (TOG)* 25.1, 19–38. [44, 66, 68].

BARGTEIL, A., SIN, F., MICHAELS, J., GOKTEKIN, T., and O'BRIEN, J. 2006. A Texture Synthesis Method for Liquid Animations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. Eurographics Association, 345–351. [62].

BÉCACHE, E., FAUQUEUX, S., and JOLY, P. 2003. Stability of Perfectly Matched Layers, Group Velocities and Anisotropic Waves. In *J. Comput. Phys.* 188.2, 399–433. [88, 98].

BERENGER, J.-P. 1994. A Perfectly Matched Layer for the Absorption of Electromagnetic Waves. In *J. Comput. Phys.* 114.2, 185–200. [29, 87, 88, 90].

BHAT, K. S., SEITZ, S. M., HODGINS, J. K., and KHOSLA, P. K. 2004. Flow-based Video Synthesis and Editing. In *ACM Transactions on Graphics (SIGGRAPH)* 23.3, 360–363. [89].

BOJSEN-HANSEN, M. 2011. A Hybrid Mesh-Grid Approach for Efficient Large Body Water Simulation. MA thesis. Aarhus University. [69].

Bojsen-Hansen, M., Li, H., and Wojtan, C. 2012. Tracking Surfaces with Evolving Topology. In *ACM Transactions on Graphics (SIGGRAPH)* 31.4, 53:1–53:10. [6, 101].

Bojsen-Hansen, M. and Wojtan, C. 2013. Liquid Surface Tracking with Error Compensation. In *ACM Transactions on Graphics (SIGGRAPH)* 32.4, 68:1–68:10. [6, 101].

Bojsen-Hansen, M. and Wojtan, C. 2016. Generalized Non-Reflecting Boundaries for Fluid Re-Simulation. In *ACM Transactions on Graphics (SIGGRAPH)* 35.4. [6].

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Levy, B. 2010. *Polygon mesh processing*. AK Peters Ltd. [57].

Botsch, M., Bommes, D., and Kobbelt, L. 2005. Mathematics of Surfaces XI: 11th IMA International Conference, Loughborough, UK, September 5-7, 2005. Proceedings. In. Springer Berlin Heidelberg. Chap. Efficient Linear System Solvers for Mesh Processing, 62–83. [17].

Bridson, R. 2008. *Fluid simulation for computer graphics*. CRC Press. [23, 64, 68, 71, 75, 95, 102].

Brochu, T., Keeler, T., and Bridson, R. 2012. Linear-Time Smoke Animation with Vortex Sheet Meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 87–95. [70, 75].

Brochu, T., Batty, C., and Bridson, R. 2010. Matching fluid simulation elements to surface geometry and topology. In *ACM Transactions on Graphics (SIGGRAPH)* 29.4, 47:1–47:9. [1, 45, 68, 69].

Brochu, T. and Bridson, R. 2009. Robust topological operations for dynamic explicit surfaces. In *SIAM Journal on Scientific Computing* 31.4, 2472–2493. [45, 52, 68, 82].

Campen, M. and Kobbelt, L. 2010. Exact and Robust (Self-) Intersections for Polygonal Meshes. In *Computer Graphics Forum (Eurographics)* 29.2, 397–406. [45].

Cantarella, J., DeTurck, D., and Gluck, H. 2002. Vector Calculus and the Topology of Domains in 3-Space. In *The American Mathematical Monthly* 109.5, 409–442. [22].

Chang, W., Li, H., Mitra, N. J., Pauly, M., and Wand, M. 2010. Geometric Registration for Deformable Shapes. In *Eurographics 2010: Tutorial Notes*. [43].

Chern, A., Knöppel, F., Pinkall, U., Schröder, P., and Weimann, S. 2016. Schrödinger's Smoke. In *ACM Transactions on Graphics (SIGGRAPH)* 35.4. [2].

Crane, K., Pinkall, U., and Schröder, P. 2013. Robust Fairing via Conformal Curvature Flow. In *ACM Transactions on Graphics (SIGGRAPH)* 32.4, 61:1–61:10. [36].

Da, F., Batty, C., and Grinspun, E. 2014. Multimaterial Mesh-based Surface Tracking. In *ACM Transactions on Graphics (SIGGRAPH)* 33.4, 112:1–112:11. [1].

Da, F., Hahn, D., Batty, C., Wojtan, C., and Grinspun, E. 2016. Surface-Only Liquids. In *ACM Transactions on Graphics (SIGGRAPH)* 35.4. [1, 101].

De Witt, T., Lessig, C., and Fiume, E. 2012. Fluid Simulation Using Laplacian Eigenfunctions. In *ACM Transactions on Graphics* 31.1, 10:1–10:11. [1].

Dinh, H., Yezzi, A., Turk, G., et al. 2005. Texture Transfer during Shape Transformation. In *ACM Transactions on Graphics (TOG)* 24.2, 289–310. [45].

Du, J., Fix, B., Glimm, J., Jia, X., Li, X., Li, Y., and Wu, L. 2006. A simple package for front tracking. In *Journal of Computational Physics* 213.2, 613–628. [45].

Edwards, E. and Bridson, R. 2014. Detailed Water with Coarse Grids: Combining Surface Meshes and Adaptive Discontinuous Galerkin. In *ACM Transactions on Graphics (SIGGRAPH)* 33.4, 136:1–136:9. [101].

Elcott, S., Tong, Y., Kanso, E., Schröder, P., and Desbrun, M. 2007. Stable, Circulation-preserving, Simplicial Fluids. In *ACM Transactions on Graphics (SIGGRAPH)* 26.1. [1].

Enright, D., Marschner, S., and Fedkiw, R. 2002. Animation and rendering of complex water surfaces. In *ACM Transactions on Graphics (SIGGRAPH)* 21.3, 736–744. [44, 68].

Enright, D., Nguyen, D., Gibou, F., and Fedkiw, R. 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proceedings of FEDSM*. Vol. 3, 4th. [68].

Fedkiw, R., Stam, J., and Jensen, H. W. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 15–22. [88].

Fujisawa, M., Mandachi, Y., and Miura, K. T. 2013. Calculation of Velocity on an Implicit Surface by Curvature Invariance. In *Journal of Information Processing* 21.4, 674–680. [101].

Gagnon, J., Dagenais, F., and Paquette, E. 2016. Dynamic lapped texture for fluid simulations. In *The Visual Computer* 32.6, 901–909. [101].

Goktekin, T., Bargteil, A., and O'Brien, J. 2004. A method for animating viscoelastic fluids. In *ACM Transactions on Graphics (SIGGRAPH)* 23.3, 463–468. [2, 55, 66, 68].

Goldade, R., Batty, C., and Wojtan, C. 2016. A Practical Method for High-Resolution Embedded Liquid Surfaces. In *Computer Graphics Forum (Eurographics)* 35.2, 233–242. [101].

Gresho, P. M. and Sani, R. L. 1987. On pressure boundary conditions for the incompressible Navier-Stokes equations. In *International Journal for Numerical Methods in Fluids* 7.10, 1111–1145. [21].

Hagstrom, T., Goodrich, J., Nazarov, I., and Dodson, C. 2005. High-order methods and boundary conditions for simulating subsonic flows. In *Proceedings of the 11th AIAA/CEAS Aeroacoustics Conference*. [88].

Heo, N. and Ko, H.-S. 2010. Detail-preserving fully-Eulerian interface tracking framework. In *ACM Transactions on Graphics (SIGGRAPH Asia)* 29.6, 176:1–176:8. [66, 68].

Hirt, C. and Nichols, B. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. In *Journal of computational physics* 39.1, 201–225. [68].

Hong, J.-M. and Kim, C.-H. 2005. Discontinuous fluids. In *ACM Transactions on Graphics (SIGGRAPH)* 24.3, 915–920. [69].

Hu, F. Q. 2001. A Stable, Perfectly Matched Layer for Linearized Euler Equations in Unsplit Physical Variables. In *J. Comput. Phys.* 173.2, 455–480. [88, 98].

Hu, F. Q. 2006. On the construction of PML absorbing boundary condition for the non-linear Euler equations. In *AIAA paper* 798, 2006. [88, 92, 93, 98].

Hu, F. Q., Li, X., and Lin, D. 2008. Absorbing boundary conditions for nonlinear Euler and Navier–Stokes equations based on the perfectly matched layer technique. In *Journal of Computational Physics* 227.9, 4398–4424. [35, 88, 92].

Hu, F., Hussaini, M., and Manthey, J. 1996. Low-Dissipation and Low-Dispersion Runge-Kutta Schemes for Computational Acoustics. In *J. Comput. Phys.* 124.1, 177–191. [88].

Jiao, X. 2007. Face offsetting: A unified approach for explicit moving interfaces. In *Journal of computational physics* 220.2, 612–625. [52].

Johnson, S. G. 2010a. *Notes on perfectly matched layers (PMLs)*. [28].

Johnson, S. G. 2010b. *Notes on the algebraic structure of wave equations*. [28, 30].

Kagaya, M., Brendel, W., Deng, Q., Kesterson, T., Todorovic, S., Neill, P. J., and Zhang, E. 2011. Video Painting with Space-Time-Varying Style Parameters. In *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 17 [1], 74–87. [62].

Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J. 2007. Simulation of bubbles in foam with the volume control method. In *ACM Transactions on Graphics (SIGGRAPH)* 26.3, 98:1–98:10. [69].

Kim, B., Liu, Y., Llamas, I., and Rossignac, J. 2007. Advections with Significantly Reduced Dissipation and Diffusion. In *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 13, 135–144. [61].

Kim, D., Lee, S. W., Song, O.-y., and Ko, H.-S. 2012. Baroclinic Turbulence with Varying Density and Temperature. In *IEEE Transactions on Visualization and Computer Graphics* 18, 1488–1495. [70].

Kim, D., Song, O.-y., and Ko, H.-S. 2009. Stretching and Wiggling Liquids. In *ACM Transactions on Graphics (SIGGRAPH Asia)* 28.5, 120:1–120:7. [2, 26, 66, 69, 70, 75, 78].

Kim, D., Song, O.-y., and Ko, H.-S. 2010. A Practical Simulation of Dispersed Bubble Flow. In *ACM Transactions on Graphics (SIGGRAPH)* 29.4, 70:1–70:5. [2].

Kim, T. and Delaney, J. 2013. Subspace fluid re-simulation. In *ACM Transactions on Graphics (SIGGRAPH)* 32.4, 62:1–62:9. [89].

Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M., and Lin, M. 2007. Texturing Fluids. In *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 13, 939–952. [62].

Lentine, M., Zheng, W., and Fedkiw, R. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM Transactions on Graphics (SIGGRAPH)* 29.4, 114:1–114:9. [66].

Li, H., Adams, B., Guibas, L. J., and Pauly, M. 2009. Robust Single-View Geometry And Motion Reconstruction. In *ACM Transactions on Graphics (TOG)* 28.5, 175:1–175:10. [43, 44, 48].

Li, H., Luo, L., Vlasic, D., Peers, P., Popovi, J., Pauly, M., and Rusinkiewicz, S. 2012. Temporally Coherent Completion of Dynamic Shapes. In *ACM Transactions on Graphics (TOG)* 31.1, 2:1–2:11. [44].

Losasso, F., Gibou, F., and Fedkiw, R. 2004. Simulating Water and Smoke with an Octree Data Structure. In *ACM Transactions on Graphics (SIGGRAPH)* 23.3, 457–462. [69].

Madsen, K., Nielsen, H. B., and Tingleff, O. 2004. *Methods for Non-Linear Least Squares Problems (2nd ed.)* Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby. [8].

Mäntylä, M. 1987. *An Introduction to Solid Modeling*. Computer Science Press, Inc. [36].

McAdams, A., Sifakis, E., and Teran, J. 2010. A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 65–74. [66].

McNamara, A., Treuille, A., Popovi, Z., and Stam, J. 2004. Fluid Control Using the Adjoint Method. In *ACM Transactions on Graphics (SIGGRAPH)* 23.3, 449–456. [1, 89].

Mitra, N. J., Flory, S., Ovsjanikov, M., Gelfand, N., Guibas, L., and Pottmann, H. 2007. Dynamic Geometry Registration. In *Proceedings of the fifth Eurographics Symposium on Geometry Processing (SGP)*. Eurographics Association, 173–182. [43].

Mullen, P., Crane, K., Pavlov, D., Tong, Y., and Desbrun, M. 2009. Energy-preserving Integrators for Fluid Animation. In *ACM Transactions on Graphics (SIGGRAPH)* 28.3, 38:1–38:8. [1].

Müller, M. 2009. Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. ACM, 237–245. [44, 55].

Museth, K. 2013. VDB: High-Resolution Sparse Volumes With Dynamic Topology. In *ACM Transactions on Graphics* 32.3, 27:1–27:22. [38, 78, 96].

Nielsen, M. B. and Bridson, R. 2011. Guide Shapes for High Resolution Naturalistic Liquid Simulation. In *ACM Transactions on Graphics (SIGGRAPH)* 30.4, 83:1–83:8. [2, 89].

Osher, S. and Fedkiw, R. 2003. *Level set methods and dynamic implicit surfaces*. Vol. 153. Springer. [2, 37, 44, 57, 64, 68].

Osher, S. and Fedkiw, R. 2006. *Level set methods and dynamic implicit surfaces*. Vol. 153. Springer Science & Business Media. [94, 95].

Palais, R. S. 2000. *An Introduction to Wave Equations and Solitons*. [12].

Pan, Z., Huang, J., Tong, Y., Zheng, C., and Bao, H. 2013. Interactive Localized Liquid Motion Editing. In *ACM Transactions on Graphics (SIGGRAPH Asia)* 32.6. [1, 89].

Park, S. I. and Kim, M. J. 2005. Vortex fluid for gaseous phenomena. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 261–270. [70].

Pfaff, T., Thuerey, N., Selle, A., and Gross, M. 2009. Synthetic turbulence using artificial boundary layers. In *ACM Transactions on Graphics (SIGGRAPH Asia)* 28.5, 121:1–121:10. [70].

Pfaff, T., Thuerey, N., and Gross, M. 2012. Lagrangian vortex sheets for animating fluids. In *ACM Transactions on Graphics (SIGGRAPH)* 31.4, 112:1–112:8. [26, 27, 70, 75, 76, 79].

Pighin, F. and Lewis, J. P. 2007. Practical Least-squares for Computer Graphics. In *ACM SIGGRAPH 2007 Courses*. [8].

Pighin, F., Cohen, J. M., and Shah, M. 2004. Modeling and editing flows using advected radial basis functions. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 223–232. [89].

Pons, J. and Boissonnat, J. 2007. Delaunay Deformable Models: Topology-Adaptive Meshes Based on the Restricted Delaunay Triangulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1–8. [45].

Pozrikidis, C. 2000. Theoretical and computational aspects of the self-induced motion of three-dimensional vortex sheets. In *Journal of Fluid Mechanics* 425, 335–366. [28, 75].

Raveendran, K., Thuerey, N., Wojtan, C., and Turk, G. 2012. Controlling Liquids Using Meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 255–264. [1].

Raveendran, K., Wojtan, C., Thürey, N., and Turk, G. 2014. Blending Liquids. In *ACM Transactions on Graphics (SIGGRAPH)* 33.4, 137:1–137:10. [2, 89].

Rossignac, J., Safonova, A., and Szymczak, A. 2001. 3D Compression Made Simple: Edgebreaker on a Corner-Table. In *2001 International Conference on Shape Modeling and Applications (SMI 2001), 7-11 May 2001, Genoa, Italy*, 278. [36].

Selle, A., Rasmussen, N., and Fedkiw, R. 2005. A vortex particle method for smoke, water and explosions. In *ACM Transactions on Graphics (SIGGRAPH)* 24.3, 910–914. [26, 70].

Sharf, A., Alcantara, D. A., Lewiner, T., Greif, C., Sheffer, A., Amenta, N., and Cohen-Or, D. 2008. Space-time Surface Reconstruction using Incompressible Flow. In *ACM Transactions on Graphics (TOG)* 27.5, 110:1–110:10. [43].

Shewchuk, J. R. 1994. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. Pittsburgh, PA, USA. [17].

Shi, L. and Yu, Y. 2005. Taming Liquids for Rapidly Changing Targets. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 229–236. [89].

Side Effects Software. 2016. *Houdini*. http://sidefx.com. [94].

Söderström, A., Karlsson, M., and Museth, K. 2010. A PML-based Nonreflective Boundary for Free Surface Fluid Animation. In *ACM Transactions on Graphics (TOG)* 29.5, 136:1–136:17. [87, 89–91, 95].

Söderström, A. and Museth, K. 2009. Non-reflective Boundary Conditions for Incompressible Free Surface Fluids. In *SIGGRAPH 2009: Talks*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 4:1–4:1. isbn: 978-1-60558-834-6. [35, 89].

Srinivasan, R. and Malkawi, A. 2007. Adaptive Localization Method: An Approach to Real Time Airflow Simulation and Immersive Visualization. In *Proceedings of the International Conference on Computer Graphics and Vision (GraphiCon)*. [90].

Stam, J. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 121–128. [88, 103].

Stam, J. and Schmidt, R. 2011. On the velocity of an implicit surface. In *ACM Transactions on Graphics (TOG)* 30.3, 21:1–21:7. [42, 52, 61, 101].

Stock, M., Dahm, W., and Tryggvason, G. 2008. Impact of a vortex ring on a density interface using a regularized inviscid vortex sheet method. In *Journal of Computational Physics* 227.21, 9021–9043. [79].

Subramanian, R. S. 2015. *Boundary Conditions in Fluid Mechanics*. [23].

Sumner, N., Hoon, S., Geiger, W., Marino, S., Rasmussen, N., and Fedkiw, R. 2003. Melting a terminatrix. In *ACM SIGGRAPH 2003 Sketches & Applications*. ACM. [57].

Sümuth, J., Winter, M., and Greiner, G. 2008. Reconstructing Animated Meshes from Time-Varying Point Clouds. In *Computer Graphics Forum (SGP)* 27.5, 1469–1476. [43].

Tessendorf, J. 2004. Simulating Ocean Waves. In *ACM SIGGRAPH 2004 Courses*. [2].

Tevs, A., Berner, A., Wand, M., Ihrke, I., Bokeloh, M., Kerber, J., and Seidel, H.-P. 2012. Animation Cartography - Intrinsic Reconstruction of Shape and Motion. In *ACM Transactions on Graphics (TOG)* 31.2, 12:1–12:15. [43].

Thürey, N., Keiser, R., Ruede, U., and Pauly, M. 2006. Detail-Preserving Fluid Control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 7–12. [89].

Thürey, N., Wojtan, C., Gross, M., and Turk, G. 2010. A Multiscale Approach to Mesh-based Surface Tension Flows. In *ACM Transactions on Graphics (SIGGRAPH)* 29.4, 48:1–48:10. [2, 57, 69].

Tong, Y., Lombeyda, S., Hirani, A. N., and Desbrun, M. 2003. Discrete Multiscale Vector Field Decomposition. In *ACM Transactions on Graphics (SIGGRAPH)* 22.3, 445–452. [22].

Treuille, A., Lewis, A., and Popovi, Z. 2006. Model Reduction for Real-time Fluids. In *ACM Transactions on Graphics (SIGGRAPH)* 25.3, 826–834. [1].

Wand, M., Adams, B., Ovsjanikov, M., Berner, A., Bokeloh, M., Jenke, P., Guibas, L., Seidel, H.-P., and Schilling, A. 2009. Efficient Reconstruction of Nonrigid Shape and Motion from Real-Time 3D Scanner Data. In *ACM Transactions on Graphics (TOG)* 28.2, 15:1–15:15. [43].

Wand, M., Jenke, P., Huang, Q.-X., Bokeloh, M., Guibas, L., and Schilling, A. 2007. Reconstruction of Deforming Geometry from Time-Varying Point Clouds. In *Proceedings of the fifth Eurographics Symposium on Geometry Processing (SGP)*. Eurographics Association, 49–58. [43].

Wang, H., Liao, M., Zhang, Q., Yang, R., and Turk, G. 2009. Physically Guided Liquid Surface Modeling from Videos. In *ACM Transactions on Graphics (TOG)* 28.3, 90:1–90:11. [43].

Wiebe, M. and Houston, B. 2004. The Tar Monster: Creating a Character with Fluid Simulation. In *ACM SIGGRAPH 2004 Sketches & Applications*. ACM. [57].

Wikipedia. 2016. *Green's function — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-May-2016]. url: `https://en.wikipedia.org/w/index.php?title=Green's_function&oldid=711364956`. [12].

Williams, B. 2008. Fluid surface reconstruction from particles. MA thesis. The University Of British Columbia. [69].

Wojtan, C., Müller-Fischer, M., and Brochu, T. 2011. Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses*. [37, 47, 48, 51, 64, 68, 84].

Wojtan, C., Thürey, N., Gross, M., and Turk, G. 2009. Deforming Meshes that Split and Merge. In *ACM Transactions on Graphics (TOG)* 28.3, 76:1–76:10. [1, 37, 45, 51, 66, 68].

Wojtan, C., Thürey, N., Gross, M., and Turk, G. 2010. Physics-inspired topology changes for thin fluid features. In *ACM Transactions on Graphics (SIGGRAPH)* 29.4, 50:1–50:8. [45, 51, 69, 78, 80].

Wojtan, C. and Turk, G. 2008. Fast viscoelastic behavior with thin features. In *ACM Transactions on Graphics (SIGGRAPH)* 27.3, 47:1–47:8. [68, 69].

Wu, J.-Z. 1995. A theory of three-dimensional interfacial vorticity dynamics. In *Physics of Fluids* 7.10, 2375–2395. [28].

Yu, J. and Turk, G. 2010. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 217–225. [69].

Yu, J., Wojtan, C., Turk, G., and Yap, C. 2012. Explicit Mesh Surfaces for Particle Based Fluids. In *Computer Graphics Forum (Eurographics)* 31.2, 815–824. [53, 57, 69].

Zaharescu, A., Boyer, E., and Horaud, R. P. 2007. TransforMesh: a topology-adaptive mesh-based approach to surface evolution. In *Proceedings of the Eighth Asian Conference on Computer Vision*. Vol. II. LNCS 4844. Springer, 166–175. [45].

Zhu, B., Lee, M., Quigley, E., and Fedkiw, R. 2015. Codimensional non-Newtonian Fluids. In *ACM Trans. Graph.* 34.4, 115:1–115:9. [2].

Zhu, Y. and Bridson, R. 2005. Animating Sand as a Fluid. In *ACM Transactions on Graphics (TOG)* 24.3, 965–972. [2, 103].