Enhancing Particle Methods for Fluid Simulation in Computer Graphics

by

Hagit Schechter

B.Sc., The Hebrew University of Jerusalem

M.Sc., The Weizmann Institute of Science

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University Of British Columbia

(Vancouver)

April 2013

Abstract

We introduce novel methods for enriched Lagrangian fluid simulation in three distinct areas: smoke animation, liquid animation, and surfacing of simulation particle data. The techniques share a common goal, to efficiently enhance realism of natural-phenomena animations in particle simulation framework.

Sub-Grid Turbulence adds synthesized small scale detail to large scale smoke simulation. The transport, diffusion and spectral cascade of turbulent energy are captured, whereas they are left unresolved on a typical simulation grid.

Ghost-SPH handling of free-surface and solid-boundary conditions in Smoothed Particle Hydrodynamics liquid simulation captures realistic cohesion for the first time and avoids the spurious numerical errors of previous approaches. Ghost particles are carefully seeded in the air and solid regions near the fluid and are assigned with extrapolated fluid quantities to reach correct boundary conditions.

The Beta Mesh is a new method for reconstructing mostly temporally coherent surface meshes from particles representing the volume of a liquid. While current particle surfacing techniques address the geometrical characteristics of the surface, the focus in Beta Mesh is producing a surface which varies smoothly in time, outside of topological changes, while preserving essential geometrical properties.

Preface

All work in this thesis was done under the supervision of Dr. Robert Bridson without other collaborators besides Dr. Bridson.

Chapter 2 was published as: H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 1–7, 2008.

Chapter 3 was published as: H. Schechter and R. Bridson. Ghost SPH for animating water. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, 31(4):61:1–61:8, 2012.

The research in this thesis was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and by a scholarship from BC Innovation Council and Precarn Incorporated.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I'd like to thank my supervisor, Dr. Robert Bridson - for the sparkle in your eyes when you taught the Physics-Based Simulation course, explaining how computer graphics, math, and physics meet. Your course made me long to learn and understand this fascinating field. Whenever I had questions, on whatever subject, you always provided knowledgeable answers and smart ideas. I deeply appreciate your confidence in me, and your conviction that I could accomplish the challenges throughout my PhD. You taught me by example that wording must be as precise as math, and illustrated how much information can be condensed in one sentence. Thank you for being a role-model on how research should be performed, always keeping it free of everything except the true academic perspective.

Second and foremost too ◡̈ I'd like to thank my husband Moshe Schechter, and our children Gal and Yuval. Moshe, you were always willing to discuss physics with me, and though you often pointed out that "fluid dynamics isn't my field", you consistently suggested amazingly intuitive explanations for why physics behaves the way it does. Throughout my PhD, you supported and backed me up when I took a leave of absence due to a submission deadline, a conference trip, or making up for a sleepless night. A huge thank you to Gal and Yuval! You were always enthusiastic about watching my animations, and ready to learn what they actually do, and why it's interesting. You gave me much appreciated advice on which colors look better, and took videos of natural phenomena that supported my submissions. And you were always nice enough to forgive me when I was sometimes too busy at work. Moshe, Gal, and Yuval, my dearest ones, you were always there for me, and without you I could never have reached the top of this mountain!

Thanks to Dr. Ian Mitchell and Dr. Wolfgang Heidrich for agreeing to be on

my committee. I deeply appreciate the very useful discussions with you, and your excellent questions, advice, and tips. I am grateful for your reviewing my text and offering useful comments, and for always being so kind and pleasant to interact with.

Thanks to the Computer Graphics professors at UBC Imager lab: Dr. Robert Bridson, Dr. Wolfgang Heidrich, Dr. Dinesh Pai, Dr. Alla Sheffer, and Dr. Michiel van de Panne, for leading an atmosphere of excellence and cooperation. It has been a pleasure to be part of such a productive environment.

Warm thanks to my colleagues in Robert Bridson's group along my PhD years: Brent, Christopher, David Y., David W., Dinos, Essex, Ivis, Landon, Mike, and Tyson. I really appreciate the way you shared your passion for computer graphics with me, the very enjoyable SPIFF hours, and the fact that you were always glad to discuss and share research ideas.

Last but not least ... Chen Greif, you were a real supporter during my PhD years. I'm truly grateful for your wise advice, and the tons of encouragement you gave me during so many supportive discussions. Thanks to Danny Cohen-Or for your consistently practical approach, for listening to my practice talks, and for suggesting useful writing tips. And to Alla Sheffer, thanks for introducing me to the computer graphics world and for your sincere care over the years.

Dedication

To the memory of my dad.

Chapter 1

Introduction

1.1 Background

The science of fluids in motion is an exciting field which affects human life in many ways. The cars, boats and airplanes that we use for transportation, the air that we breathe, and the daily weather forecast, are just a few examples of the influence of fluid flow in our everyday lives. A significant body of research around fluid simulation has accumulated since the development of powerful computers in the 20th century, significantly enabling fields such as aerodynamics and hydrodynamics, which in turn have improved the efficiency of transportation and facilitated the process of globalization.

Adapting or inventing new fluid simulation techniques has taken on increasing importance for computer graphics applications in the last two decades. A demand for realistically animated phenomena such as liquids, smoke, and fire in feature films drove the visual effects industry and the academic research community to seek better solutions. The computer game industry has also demanded improved techniques for realistic gaming experiences in real time. Alongside those trends, the improvement of commodity graphics and computer hardware has made increasingly complicated interactive applications and off-line computation tasks feasible. The characteristics of the film industry and the gaming industry applications are still quite different. In the film industry the simulated fluids have to match reality so that an observer cannot distinguish between real scenarios and simulated envi-

ronments. The gaming industry, on the other hand, also aims at maximum reality but must often compromise in favor of the algorithms' performance. Hardware advancements may diminish those differences in the future, but it will likely be enhanced algorithms which will enable more realistic, richer online game animations.

There are two approaches to tracking fluid motion, the Lagrangian viewpoint and the Eulerian viewpoint. The Lagrangian approach samples the continuum with particles that are assigned with properties such as mass, velocity, or temperature, that move in the domain determined by the governing equations' forces. In the Eulerian approach, fluid quantities are measured in fixed points in space which get updated by the fluid flow through these points. In this thesis we will focus on pure Lagrangian and on hybrid Lagrangian-Eulerian techniques. See Bridson's book [17] for a good practical overview of the basic methods of fluid simulation and their implementation in computer graphic applications.

The use of particle algorithms for realistic fluid animation has been steadily growing in the last decade. There are various advantages to using particles. Particle dynamics use a very robust and simple treatment of advection: the motion of the flow is precisely the motion of the particles. Complicated geometrical structures can be supported with just a few simple primitives, and, as a result, detail loss is avoided. In addition, particle techniques are often easier to implement than other techniques. The use of particle systems in computer graphics goes back all the way to Reeves' pioneering work [59], which illustrated how complicated concepts such as fire can be animated using simple primitives: particles. Reeves' particle system was used to implement the wall of fire element in the "Star Trek II: The Wrath of Khan" film (1982). Blinn's paper [12] is another significant early particles work that demonstrated surface reconstruction from native spherical particle geometry using implicit surfaces. Various later surface reconstruction techniques followed Blinn's idea and further improved upon it.

We highlight here two particle methods that are commonly used in computer graphics and are particularly relevant to this thesis. The *Particle In Cell* (PIC) approach, introduced by Harlow [35], uses a hybrid grid-particles approach. All forces, e.g. pressure gradient force, are implemented on the grid just like a fully Eulerian solver, whereas advection is done purely on particles using interpolated

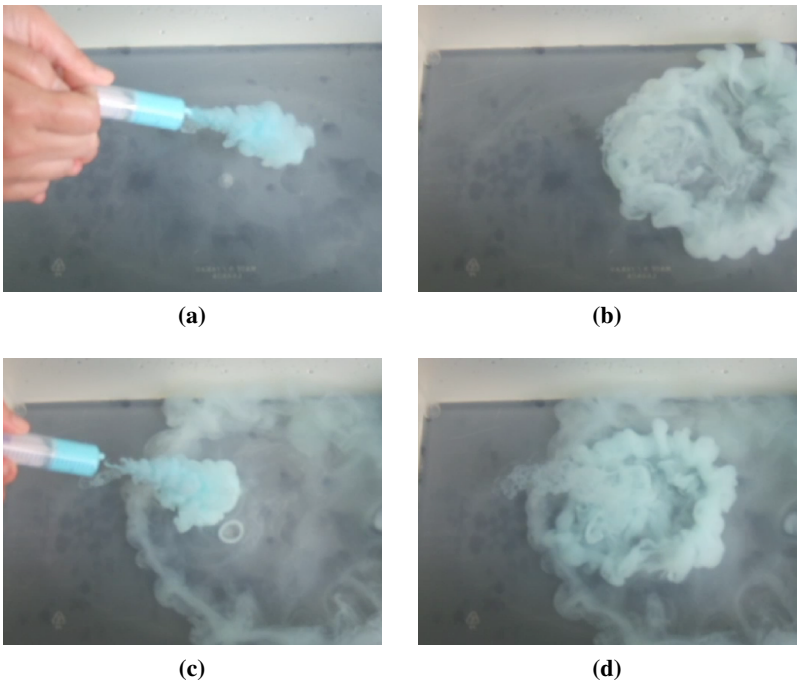
grid velocities. The resulting velocities then get transferred from the particles back to the grid. PIC suffered from excessive numerical dissipation, which was cured by the Fluid Implicit Particle (FLIP) method introduced by Brackbill and Ruppel [15]. In FLIP, particle velocities instead get updated with the change in grid velocity from the previous step. Zhu and Bridson [77] adapted FLIP to incompressible flow, and further extended it to model sand for animation. The *Smoothed Particle Hydrodynamics* (SPH) is a Lagrangian, mesh free method, introduced independently by Lucy [45] and Gingold and Monaghan [34]. The governing equations' quantities on the particles are approximated using a smoothing kernel over neighboring particles. See Monaghan's SPH review [48] for further details. Desbrun and Cani [24] were the first to introduce SPH to computer graphics for viscous material animation. Later Müller et al. [49] used SPH for interactive water simulation, followed by many other works in computer graphics.

1.2 Challenges in Computer Graphics Applications

Several major problems exist when simulating fluids for graphics applications.

Efficiently achieving small scale turbulence detail: Turbulent flows can be observed in our everyday surroundings, whether it be smoke from a chimney, buffeting of a strong wind, or water flowing in a waterfall. The flow is characterized by motions of many scales, and is always irregular, unsteady, and seemingly chaotic. Direct Numerical Simulation (DNS) methods resolve the entire range of turbulent length scales but are extremely expensive and impractical to be used for large domains. Shinya and Fournier [63] and Stam and Fiume [70] introduced to graphics perhaps the simplest physically reasonable turbulence model, the Kolmogorov "5/3-law", which injects kinetic energy of order $k^{-\frac{5}{3}}$ where k is the spatial Fourier frequency. The stochastic sub-scale modes that were added to the flow provided the desired additional detail, but lacked the capture of the time evolution of turbulence that is visually important.

Effective boundary conditions for pure particle methods: Proper treatment of the fluid at the boundaries is extremely important to produce visually realistic

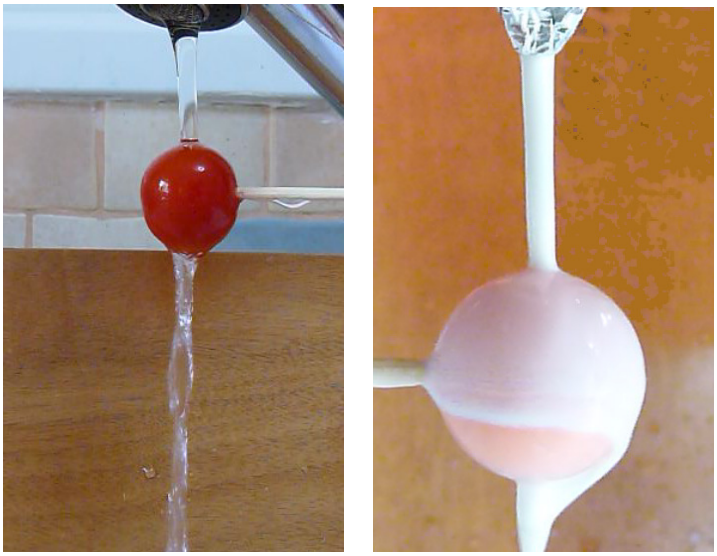


(a) **(b)** **(c)** **(d)**

Figure 1.1: Turbulent Motion in Water Tank (Real Footage). Colored milk injected into a water tank showing multiple scales and time evolution of turbulent motion. Alphabetic order provides chronological non-consecutive steps.

animation. Several types of boundaries exist: free surface (a common, practical, simplified model for the interface of liquids with air), solid walls (the interface with solids), and two-phase flow interface (the interface with another fully-simulated fluid). We will focus here on the first two types. Fedkiw [30] introduced the Ghost Fluid Method (GFM) for Eulerian methods. Fedkiw characterized quantities that are continuous across the interface (such as the normal component of velocity), quantities that are discontinuous and uncoupled (like the tangential component of velocity in inviscid flow), and other quantities that are discontinuous and coupled (e.g. the normal component of the pressure gradient across the interface in two-phase flow). Accordingly, air or solids at the fluid interface are treated like another fluid, and their grid cells are assigned with appropriate values required for correct interface modeling. In particle methods however, the solids or air are generally

not sampled with particles, which prohibits using the GFM mechanism to enforce the correct boundary conditions. In the SPH literature in particular, various works suggest applying direct forces at the solid boundary, e.g. [47]; these do not yield correct boundary conditions however, but rather apply repulsion at the interface to prevent solid penetration. Free surfaces impose an even bigger problem in SPH, as the air does not even have associated geometry as solids do. As a result, the most common form of SPH in graphics forms a dense shell of particles at the interface with air to compensate for the deficient density profile.



(a) (b)

Figure 1.2: Tomato Under Stream of Tap Water and Milk (Real Footage). Tomato under stream of tap water (left) and milk (right) showing cohesion of liquids to solids in real-life.

Temporally coherent surface reconstructions from particles: Reconstructing a surface around the particles is generally essential for rendering the flow, and representing the fluid surface as a mesh in particular makes it more amenable to further processing, rendering with displacement textures, motion blur, etc. The standard solution is to construct a smooth implicit surface wrapped around the particles, as in the "blobbies" [12] approach. The blobbies surface can however have

noticeable artifacts, and smooth surfaces cannot be reproduced even when the fluid surface is perfectly smooth. Major advances have been reached later by improving this technique, targeting better geometrical properties and surface smoothness. But superior geometrical properties are just part of a desirable surfacing technique — we also desire that the surface evolves smoothly in time, performs natural fluid merges and splits, and avoids any "pops" that reflect surfacing limitations. Creating surfaces that are temporally coherent has had very little attention in the literature so far.

1.3 Thesis Contributions

This thesis addresses the challenges described in Section 1.2.

Chapter 2 addresses the multi-scale and time evolution turbulence challenge. We introduce a simple turbulence model for smoke animation, qualitatively capturing the transport, diffusion, and spectral cascade of turbulent energy unresolved on a typical simulation grid. We track the mean kinetic energy per octave of turbulence in each grid cell, and a novel "net rotation" variable for modeling the self-advection of turbulent eddies. These additions to a standard fluid solver drive a procedural post-process, layering plausible dynamically evolving turbulent details on top of the large-scale simulated motion. Finally, to make the most of the simulation grid before jumping to procedural sub-grid models, we propose a new multistep predictor to alleviate the nonphysical dissipation of angular momentum in standard graphics fluid solvers. This paper was presented at the 2008 Symposium on Computer Animation [60].

Chapter 3 addresses the boundary conditions challenge in SPH. We propose a new ghost fluid approach for free surface and solid boundary conditions in Smoothed Particle Hydrodynamics (SPH) liquid simulations. Prior methods either suffer from a spurious numerical surface tension artifact or drift away from the mass conservation constraint, and do not capture realistic cohesion of liquid to solids. Our Ghost SPH scheme resolves this with a new particle sampling algorithm to cre-

ate a narrow layer of ghost particles in the surrounding air and solid, with careful extrapolation and treatment of fluid variables to reflect the boundary conditions. We also provide a new, simpler form of artificial viscosity based on XSPH. Examples demonstrate how the new approach captures real liquid behaviour previously unattainable by SPH with very little extra cost. The paper was presented at the 2012 SIGGRAPH conference on Computer Graphics and Interactive Techniques, and published in the journal ACM Transactions on Graphics [61].

Chapter 4 investigates a novel technique for temporally coherent surface reconstruction. We present a new approach — the beta mesh — for generating surfaces from animated particle data, targeting temporally coherent surface reconstruction that can also approximate smooth surfaces and capture fine details. Our beta mesh algorithm uses the union of balls as a building block to reach temporal coherence. First we construct mesh vertices from sphere intersection points, and declare faces on the spheres surface guided by connectivity intelligence derived from the alpha mesh. Then we smooth the beta vertices positions to reflect smooth surfaces, and subdivide the mesh using weighted centroids. We also highlight the strengths and weaknesses of the related alpha mesh for animation purposes, and discuss ways of leveraging its qualities. Open issues are discussed to outline what is still lacking in order to make our algorithm a ready-to-use surfacing technique. Nevertheless, we advocate using the beta mesh approach in future surface reconstruction research to benefit from its unique properties.

1.4 Thesis Structure

This thesis consists of three research topics presented in chronological order. Each chapter of the thesis is self-contained, includes its own notation, discussion, and results. Chapter 2 and Chapter 3 are brought here in their published form up to minor adjustments. A conclusion chapter highlights the main contributions in this thesis and outlines future research directions.

Chapter 2

Evolving Sub-Grid Turbulence for Smoke Animation

2.1 Contributions

Animating turbulent fluid velocity fields, from the delicate swirls of milk stirred into coffee to the violent roiling of a volcanic eruption, poses serious challenges. While the look of the large-scale components of motion are well captured by direct simulation of the fluid equations, increasing the grid resolution to capture the smallest turbulent scales hits a severe scalability problem. The usual solution of augmenting a coarse simulation with procedurally synthesized turbulent detail generally is limited by the visual implausibility of this detail: while some statistics or invariants may be matched, visually important aspects of the time evolution of turbulence are still missing.

This paper attempts to bridge the gap by introducing a turbulence model that simulates the transfer of energy in sub-grid scales, then providing a procedural method for instantiating a high resolution turbulent velocity field from this data, which can be evaluated on the fly for a marker particle pass when rendering. For each octave of sub-grid detail, in each grid cell, we evolve both a kinetic energy density E and a net rotation θ for generating the turbulence.

In addition, since full fluid simulation is generally preferred to procedural models when feasible, we address one of the chief remaining sources of nonphysical

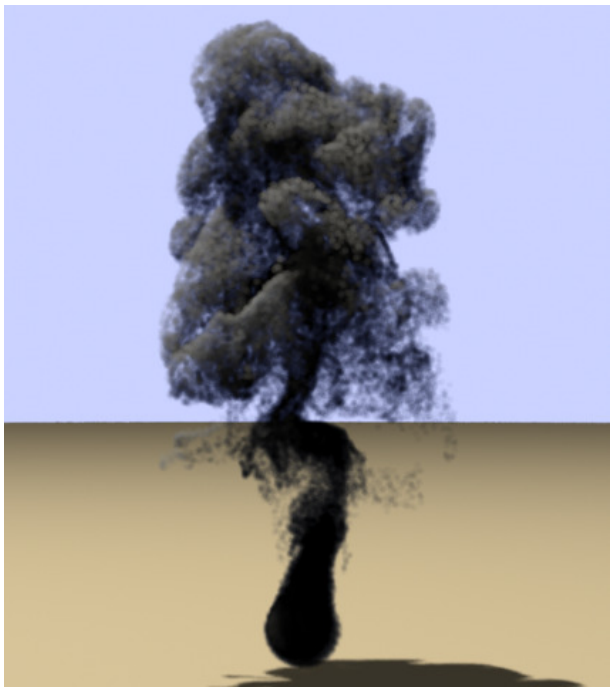


Figure 2.1: Hot Smoke Simulation. A simulation of a hot smoke source is augmented with our turbulence evolution model.

energy dissipation in graphics fluid solvers, adding a predictor step to the usual time splitting of the incompressible Euler equations. This helps make the most of a limited grid size.

2.2 Background

The field of turbulence modeling is vast; for an overview we refer readers to the recent text by Pope [56] or the earlier classic by Tennekes and Lumley [72]. For an introduction to basic fluid simulation techniques within graphics, see Bridson and Müller-Fischer's course notes [18].

There is a huge disparity in length scales for turbulent flow: e.g. in the atmosphere large-scale features may be measured in kilometres, with the smallest features in millimetres. A fine enough grid to capture everything (the idea behind the Direct Numerical Simulation approach in science) may be enormous: n^3 grid

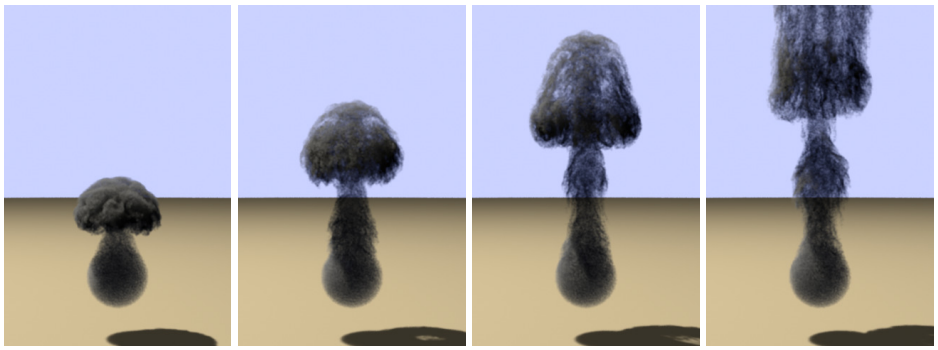


Figure 2.2: A Series of Smoke Simulation Frames. Frames from an animation with the new turbulence model.

cells with $n > 1000$. Unfortunately, turbulence fills the entire volume with fine-scale features, eliminating the benefit of adaptive grid methods, and to resolve the small scales time steps must be proportional to the grid spacing, resulting in at least $O(n^4)$ work. This severely limits the scalability of straight simulation for capturing turbulence.

However, turbulence research has developed higher level models which promise efficient simulation, based on the observation that smaller scales in turbulence quickly become isotropic and more or less statistically independent. This allows useful notions of average effect and evolution without need of resolving all details, with the chief visible actions being:

- Enhanced mixing, which when averaged over appropriate length and time scales can be interpreted as a diffusion process, with a so-called "eddy viscosity". This is why stirring milk in coffee is much more effective than letting it sit still.

- Transfer of energy from large-scale structures to smaller-scale eddies and ultimately to the smallest scales where molecular viscosity takes over to dissipate kinetic energy into heat. This is the cause of such a large range of length scales.

The simplest model that captures these, the Kolmogorov $5/3$ power law, has long been used in graphics for synthesizing velocity fields with plausible statistics. As-

suming uniform, steady, self-preserving turbulence, where the energy continuously injected at the largest scales matches the energy dissipated by viscosity at the smallest scales and an equilibrium in the intermedate scales, this provides a simple formula for the amount of energy and rate of fluctuation in each frequency band.

Many researchers have augmented fluid simulations with sub-grid turbulence models, i.e. directly simulating the large scales while estimate the effect of unresolved smaller scales. These methods separate the velocity field into a sum of the "mean flow" (the large-scale components) and a turbulent fluctuation. For the Reynolds Averaged Navier-Stokes (RANS) approach, the mean-flow averaging is taken over appropriate time scales, and for the Large Eddy Simulation (LES) approach the averaging is done with a spatial kernel such as a Gaussian. Averaging the Navier-Stokes equations gives rise to very similar equations for the mean flow, but with the addition of a "Reynolds stress" giving the effect of turbulent mixing on the mean flow—at its simplest modeled as a nonlinear viscosity. For this paper, where we try to capture the qualitative look but don't provide quantitatively accurate results, we assume numerical dissipation in the usual first-order accurate fluid solvers dominates this term and thus ignore it.

Some turbulence methods go further by tracking information about the turbulent fluctuations to produce better estimates of their effect on the mean flow. We focus in particular on the popular $k-\varepsilon$ model, originating in work by Harlow and Nakayama [36]. Here two more fields are added to the simulation, k being the kinetic energy density of the turbulent fluctations (energy per unit volume) and ε representing the rate of energy dissipation in the viscous scales. Unlike our new model, this does not explicitly track the cascade of energy from large scales to small scales, but like our model simulates the spatial transport, diffusion, and ultimate dissipation of turbulent energy.

Turning to computer graphics, Shinya and Fournier [63] and Stam and Fiume [70] used the Kolmogorov $5/3$ law and the inverse Fourier transform to generate incompressible velocity fields with plausible statistics, possibly layered on top of a large-scale flow. This was later also used by Rasmussen et al. [58] to break up smoothness in the interpolation of 3D velocity fields from simulated 2D slices. Kniss and Hart [43] demonstrated that by taking the curl of vector valued noise functions, plausible incompressible velocity fields can be created without need for

Fourier transforms; Bridson et al. [19] extended the curl-noise approach to respect solid boundaries and conveniently construct larger-scale patterns, and illustrated the attraction of spatial modulating turbulence. Most recently Kim et al. [42] combined curl-noise with a wavelet form of the $5/3$ law, extending the energy density measured in the highest octave of a simulation into a turbulent detail layer and further advecting the noise texture with the flow to capture spatial transport of turbulent flow features, with marker particles for rendering.

Neyret's work on advected textures [52] is closest in spirit to this paper. He introduces a single representative vorticity vector per simulation grid cell and per octave of turbulence, advected with the flow. The vectors are gradually blended from coarse octaves to fine octaves, and are used to rotate the gradients in flow noise [55]. The flow noise is then used to distort a texture map (e.g. a smoke-like hypertexture), with smooth regeneration of texture detail via blending of the original when total deformation has increased past a limit. We instead track (and conserve in all but the finest octave) kinetic energy, include the spatial diffusion of kinetic energy, and properly decouple the rotation of nearby gradient vectors in flow noise (so not all of the gradient vectors in one simulation cell are rotated with the same vorticity); we further combine it with curl-noise to produce velocity fields for marker particle advection in rendering.

Our predictor method for reducing nonphysical rotational dissipation in the large-scale fluid simulation also has antecedents in CFD and graphics. The chief culprits are in the treatment of the advection term in the momentum equation. Fedkiw et al. [31] observed that some of the strong numerical dissipation of the trilinear interpolation semi-Lagrangian method introduced by Stam [69] can be reduced by using a sharper Catmull-Rom-based interpolant instead; many other improvements to pure advection followed (e.g. Kim et al.'s BFECC approach [41]) culminating in Zhu and Bridson's FLIP method [77] which essentially eliminates all numerical dissipation from advection via use of particles. However, this is not the only source of dissipation: the first order time splitting used in graphics, where velocities are separately advected and then projected to be incompressible, also introduces large errors. In particular, angular momentum is not conserved even with a perfect advection step, as can readily be seen if a rigidly rotating fluid is advected by 90° in one time step: the angular velocity would be entirely transferred into a diver-

gent field, which pressure projection would subsequently zero out. In the scientific literature, predictor-corrector, multistep or Runge-Kutta methods are used in conjunction with high resolution discretizations of the advection term and small time steps to avoid this problem. However, these aren't directly applicable to the large time steps taken with semi-Lagrangian advection or FLIP in graphics.

Vorticity confinement [31] helps recover some of the lost angular momentum, by adding artificial forces to enhance spin around local maxima of the existing vorticity. Selle et al. [62] extended this with spin particles, to boost the smallest-scale vorticity present on the grid. However neither technique is applicable to the simplest case, reducing the dissipation in rigid rotation, and thus we propose a more general solution in this paper.

We also note that alternative approaches to fluid simulation, in particular vortex particle methods (e.g. [4, 33, 53, 75]), do not suffer from this dissipation. However vorticity methods have their own share of problems, particular in 3D, such as in handling free surface and solid boundary conditions, thus most work has centred around velocity/pressure methods.

2.3 Characterizing Sub-Grid Turbulence

Our goal is to simulate the average behavior of the sub-grid turbulence, leaving the details of instantiating a plausible velocity field to a post-simulation phase. In particular, we want to describe the turbulence without recourse to higher resolution grids. Note that we assume that the combination of FLIP with our new multistep time integration will capture all grid-level turbulence directly in simulation, thus we only focus on the missing sub-grid part.

We extend the $k-\varepsilon$ approach of tracking mean kinetic energy density: instead of a single total kinetic energy density per grid cell, we break it up into octaves corresponding to spatial frequency bands (similar to the usual notion of "turbulence" textures in graphics). For example, with three octaves we store three kinetic energy densities in each grid cell (i,j,k): $E^{(1)}_{ijk}$, $E^{(2)}_{ijk}$, and $E^{(3)}_{ijk}$, using E instead of the traditional k to avoid confusion with grid cell indices. The first, $E^{(1)}$, corresponds to the components with wavelengths approximately between Δx and $\frac{1}{2}\Delta x$, the $E^{(2)}$ value for wavelengths between $\frac{1}{2}\Delta x$ and $\frac{1}{4}\Delta x$, etc. For future work it might be use-

ful to even synthesize turbulence at coarser scales ill-resolved by the simulation, for example $E^{(0)}$ at wavelengths $2\Delta x$ to Δx. This allows us to track the transfer of energy in spectrum as well as space—opening the door to handling the transition from laminar to turbulent flow, or the unsteady evolution and decay of unsustained turbulence. This still has attractive scalability: to increase the apparent resolution of a simulation by a factor of 2^k our memory use only increases by $O(k)$, leading to $O(n^3k)$ total memory usage. We cannot distinguish variations in turbulent energy at sub-grid resolution, but since the turbulent energy undergoes a diffusion process, sub-grid variations shouldn't be visually significant.

We also address the issue of temporal coherence in the turbulence field. While the energy density is enough to procedurally synthesize a plausible velocity field for a single frame, we want to reflect the correlation from one frame to the next. Inspired by flow noise [55] we add an additional scalar variable $\theta_{ijk}^{(b)}$ in each grid cell for each octave, an estimate of the average amount of rotation in that region of space up to the current time caused by the turbulent components, i.e. the time integral of the magnitude of vorticity at a fixed location in space. We use this to directly control flow noise when instantiating velocities.

The details of the algorithm are described in the following sections, and a pseudocode is provided in Algorithm 1 and Algorithm 2.

2.4 Evolving Sub-Grid Turbulence

We break up the problem into plausibly evolving the kinetic energies $E_{ijk}^{(b)}$ and separately updating the θ angles.

2.4.1 Energy Evolution

Note that the large-scale flow on the grid transports with it small-scale turbulent components. Therefore, just as in the $k-\varepsilon$ model we begin with advecting the kinetic energy densities with the usual fluid variables in the simulation.

In addition, the enhanced mixing of turbulent flow is usually modeled as a nonlinear spatial diffusion term, spreading turbulent energy across space. We make a crude simplification to constant-coefficient linear diffusion instead; from a scientific viewpoint this is probably unjustifiable, yet we argue it visually captures the

qualitative effect of eddy viscosity while avoiding numerical complications caused by more accurate nonlinear models.

Finally, the nonlinear advection term in the Navier-Stokes momentum equation causes transfer of kinetic energy between different frequency bands. While this happens in both directions, it is widely modeled that the dominant effect in turbulence is the "spectral cascade" of energy from low frequencies to higher frequencies, with the majority of the energy transfer being between nearby wavelengths. We therefore include a simple transfer term from the kinetic energy in one octave to the next octave along, similar to Neyret's vorticity transfer. Again, the true energy transfer is a nonlinear process and our constant-coefficient linear simplification is scientifically invalid, but is useful as the simplest possible model that captures the qualitative visual effect of the spectral cascade.

We discretize this for a single time step on a grid as follows, with given coefficients $\alpha \geq 0$ and $\beta \geq 0$ controlling spatial diffusion and spectral cascade respectively:

- Advect the per-octave kinetic energies with the large-scale flow to get intermediate energies $\bar{E}^{(b)}_{ijk}$.
- Apply spatial diffusion and scale-to-scale transfer to get the energies at the next time step:

$$E^{(b)}_{ijk} = \bar{E}^{(b)}_{ijk} + \alpha\Delta t \begin{pmatrix} \bar{E}^{(b)}_{i+1,jk} + \bar{E}^{(b)}_{i-1,jk} \\ +\bar{E}^{(b)}_{ij+1,k} + \bar{E}^{(b)}_{ij-1,k} \\ +\bar{E}^{(b)}_{ijk+1} + \bar{E}^{(b)}_{ijk-1} \\ -6\bar{E}^{(b)}_{ijk} \end{pmatrix} + \beta\Delta t(\bar{E}^{(b-1)}_{ijk} - \bar{E}^{(b)}_{ijk}) \quad (2.1)$$

There is a time step restriction of $\Delta t < (6\alpha + \beta)^{-1}$ to guarantee that the $\bar{E}^{(b)}_{ijk}$ coefficient is non-negative, which in our examples was never particularly restrictive. Strictly speaking the α and β coefficients should take into account length scales and the energies themselves, but we choose to keep them as simple tunable constants. In our evolution equation the first octave $\bar{E}^{(1)}$ refers to a nonexistent $\bar{E}^{(0)}$: this ideally should be energy pulled from the large-scale simulation itself, but we instead simply seeded it with a spatial texture, e.g. introducing turbulent energy

Algorithm 1 Large-Scale Simulation Step

Input: FLIP grid G and particles P, number of octaves b

Output: grid velocity $\{\vec{V}_{ijk}\}$, grid energy $\{E^{(b)}_{ijk}\}$ for each octave b, grid net rotation $\{\theta^{(b)}_{ijk}\}$ for each octave b

Instantiate smoke source:
for all particle $p \in P$ in smoke source region **do**
    Inject buoyancy parameters: temperature T_p, soot S_p
    Inject first octave energy E^0_p
FLIP particles $\rightarrow$ grid:
for all particle $p \in P$ **do**
    Transfer velocity $\vec{V}_p$ to grid with predictor (Equation 2.11)
    Transfer temperature T_p and soot S_p to grid
    Transfer first octave energy E^0_p to grid
Evolve grid properties:
for all grid point $(i, j, k) \in G$ **do**
    Advance buoyancy parameters: temperature T_{ijk}, soot S_{ijk}
    Advance acceleration $\vec{a}_{ijk}$ with forces
    Advance velocity with acceleration $\vec{V}^*_{ijk} \leftarrow \vec{V}_{ijk} + \Delta t \vec{a}_{ijk}$
Pressure projection solve to apply divergence free field $\{\vec{V}^{new}_{ijk}\}$
for all octave b, grid point $(i, j, k) \in G$ **do**
    Transport energy $E^{(b)}_{ijk}$ with Equation 2.1
for all octave b, grid point $(i, j, k) \in G$ **do**
    Advance net rotation $\theta^{(b)}_{ijk}$ with Equation 2.3
FLIP grid $\rightarrow$ particles:
for all grid point $(i, j, k) \in G$ **do**
    Transfer velocity $\{\vec{V}^{new}_{ijk}\}$ to particles $\{\vec{V}^{new}_p\}$ with predictor (Equation 2.11)
Advect:
for all particle $p \in P$ **do**
    Update particle position $\vec{X}^{new}_p$ using RK3

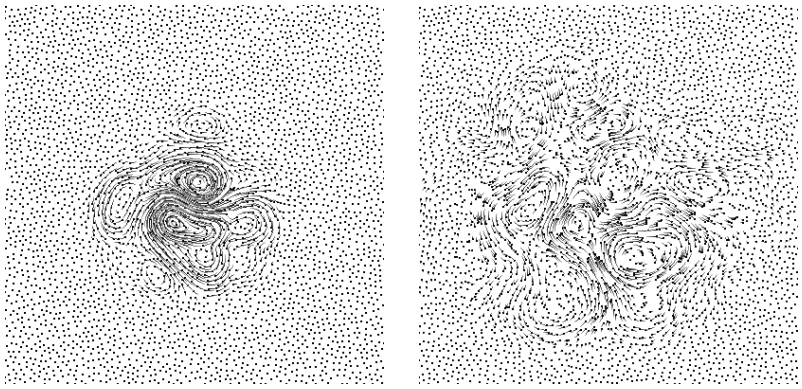


Figure 2.3: 2D Energy Transport. Left: an initial disturbance in the lowest octave only (large circles). Right: snapshot later in time, demonstrating diffusion in space (motion on the right is spread beyond the initial area), and energy cascade to higher octaves (mixing of large circles and small circles).

just in the source of a smoke plume. At the last octave, we optionally add another term $\gamma \Delta t \bar{E}^{(b)}$ with $0 \leq \gamma \leq \beta$ to partially cancel out the loss of energy to untracked octaves. See figure 2.3 for an example of the effect of both diffusion and spectral cascade.

We underscore that this evolution doesn't influence the large-scale motion at all, under the assumption that errors in the large-scale fluid simulation will dominate the effect of sub-grid turbulence. This decoupling has the attractive side-effect that turbulence evolution can be done as a post-process: once a suitable large-scale simulation has been found, different initial conditions and parameters for turbulence evolution can be tried out very efficiently.

We ran examples of the new turbulence model coupled with a FLIP simulation of smoke [77], with the standard addition of heat and buoyancy forces to the fluid solver. We advected the turbulent kinetic energy variables with the FLIP particles, transferring them to the grid, applying diffusion and spectral cascade on the grid, and transferring the result back to the particles in the usual FLIP way.

2.4.2 Net Rotation Evolution

The net rotation $\theta_{ijk}^{(b)}$ is an estimate of the time integral of the magnitude of vorticity stemming from the b'th octave of turbulence in grid cell (i, j, k). This is *not* a quantity to be advected: it's an Eulerian history variable. We also emphasize this is just a scalar, unlike the vector-valued vorticity, since we are using this to estimate average rotation over a region of space (containing many differently-oriented vorticities), analagous to our use of scalar kinetic energy density rather than mean velocity.

At every time step of the turbulence evolution simulation, we increment $\theta_{ijk}^{(b)}$ by an estimate of the average magnitude of the vorticities in the b'th octave, which we assume are proportional to the mean speed (available from the kinetic energy density) divided by the wave length:

$$\|\omega\|_{ijk}^{(b)} \sim \frac{\sqrt{2E_{ijk}^{(b)}/\rho}}{2^{-b}\Delta x} \tag{2.2}$$

This is dimensionally correct and consistent with the observation that turbulence typically remains isotropic. Introducing a constant of proportionality δ we use the following update:

$$\theta_{ijk}^{(b)} \leftarrow \theta_{ijk}^{(b)} + \delta\Delta t \frac{\sqrt{2E_{ijk}^{(b)}/\rho}}{2^{-b}\Delta x} \tag{2.3}$$

For the full effect we should also introduce a $\theta^{(0)}$ variable that is updated by the magnitude of vorticity in the large-scale simulation, but we have not yet experimented with this.

A pseudocode for the large-scale simulation is provided in Algorithm 1. Section 2.7 describes our velocity predictor that helps reducing angular dissipation during the large-scale simulation.

2.5 Instantiating Small-Scale Velocity Fields

Once we have completed a large-scale fluid simulation and then turbulence evolution, we generate a plausible total velocity field by interpolating from the large-scale velocity grid and adding to it the curl of a vector potential derived from the

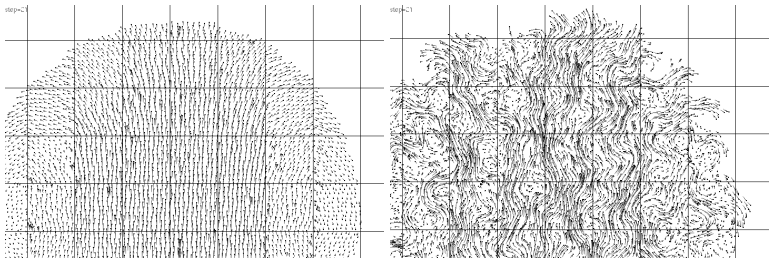


Figure 2.4: 2D Large-Scale vs. Small-Scale. On the left is the large-scale velocity field interpolated from the displayed grid (unrealistically smooth for illustration purposes). On the right we add sub-grid turbulent scales to get the total velocity field.

turbulence quantities, as in the curl-noise approach of Kniss and Hart [43] and Bridson et al. [19]: see figure 2.4.

Perlin and Neyret's flow noise [55] was originally designed to provide fluid-like textures which animate in time with a pseudo-advection quality (unlike regular time-animated noise which smoothly changes without coherent flow structure): the gradient vectors underling Perlin noise are rotated in time as if by fluid vorticity. Neyret further used this for deforming other textures in his advected texture work [52], and Bridson et al. [19] suggested it is well suited for use in curl-noise to generate velocity fields featuring vortices (corresponding to peaks in the noise functions) which naturally swirl around and interact with other vortices rather than simply smoothly appearing and disappearing. We therefore adopt flow noise to build the vector potential $\vec{\psi}$ for the turbulent velocity contribution; however, since vorticities in turbulence should be uniformly distributed due to isotropy, we modify Neyret's scheme which only used a single vorticity per grid cell per octave.

We begin with an auxiliary array of 128 pairs of uniformly randomly selected orthogonal unit three-dimensional vectors, $\hat{p}$ and $\hat{q}$, which will be used to parameterize a plane of rotation at each point in the lattice. A lattice point (i, j, k) is mapped to the array using a hash function $hash(i, j, k)$ discussed later in this section. To evaluate a given octave of the noise at a point, we look up the rotated gradients at the corners of the lattice cell containing the point. This lattice, for octave b, has a spacing of $2^{-b+1}\Delta x$ compared to the Δx spacing of the simulation

grid. Hence while the auxiliary array is fixed, each octave accesses it in its own frequency, higher octave uses smaller wavelength and hence higher frequency. At each lattice point we use the hash to look up a pair of orthogonal unit vectors, $\hat{p}$ and $\hat{q}$, trilinearly interpolate the net rotation $\hat{\theta}$ in this octave from the simulation using spline interpolation weights for smooth transitions, see Perlin [54] for full details on this gradient interpolation scheme. Finally we form a rotated gradient vector as:

$$\vec{g} = \cos\hat{\theta}\hat{p} + \sin\hat{\theta}\hat{q} \tag{2.4}$$

One of the most expensive parts of this calculation is the trigonometic function evaluation, but after all the modeling errors in the system it's not crucial that these be accurate; we substituted the following cubic spline

$$\sin\theta \approx 12\sqrt{3}t\left(t-\frac{1}{2}\right)(t-1) \tag{2.5}$$

where t is the fractional part of $\theta/(2\pi)$, and similarly approximated $\cos(\theta) = \sin(\theta+\frac{1}{2}\pi)$. We found this was sufficiently accurate and much faster. (Note that this spline has zeros at the correct roots of sine and attains a max and min of ± 1 as expected.) Finally, we evaluate three independent noise functions to get a vector noise value $\vec{N}^{(b)}(\vec{x},t)$.

We also trilinearly interpolate the kinetic energy density in an octave at any point in space from the values on the simulation grid and use spline interpolation weights for smooth transitions, similar to the rotation angle interpolation scheme. We then estimate an appropriate amplitude for modulating the noise:

$$A^{(b)}(\vec{x},t) = C2^{-b}\Delta x\sqrt{2E^{(b)}(\vec{x},t)/\rho} \tag{2.6}$$

The user-defined constant C should be approximately 1, so that the actual kinetic energy density of the velocity field below will match up to the stored $E^{(b)}$. Adding up the octaves gives the vector potential ψ:

$$\vec{\psi}(\vec{x},t) = \sum_b A^{(b)}(\vec{x},t)\vec{N}^{(b)}(\vec{x},t) \tag{2.7}$$

Further modifications to respect solid boundaries can be made per Bridson et al. [19].

The turbulent part of the total velocity field is the curl of $\vec{\psi}$.

Algorithm 2 Small-Scale Simulation Step

Input: grid velocity $\{\vec{V}_{ijk}\}$, grid energy $\{E_{ijk}^{(b)}\}$ for each octave b, grid net rotation $\{\theta_{ijk}^{(b)}\}$ for each octave b, sub-grid particles $P = \{p_i\}$ and positions $\{\vec{x}_i\}$
Output: new sub-grid particles P' and positions $\{\vec{x}_i^{new}\}$

Add particles in smoke source region $P' \leftarrow P \cup \{p_j^{new}\}$
for all sub-grid particle $p_i \in P'$ **do**
    Compute large-scale velocity $\vec{v}_i^l$ at $\vec{x}_i$ from grid velocities $\{\vec{V}_{ijk}\}$
    Compute small-scale velocity $\vec{v}_i^s \leftarrow \nabla \times \psi(\vec{x}_i)$ using Algorithm 3
    Compute particle new position $\vec{x}_i^{new} \leftarrow \vec{x}_i + \Delta t(\vec{v}_i^l + \vec{v}_i^s)$

Algorithm 3 Compute Potential at Position $\vec{x}$

Input: position $\vec{x} = (x,y,z)$, array of basis vectors $basis[128][2]$, grid net rotation $\{\theta_{ijk}^{(b)}\}$ for each octave b, grid energy $\{E_{ijk}^{(b)}\}$ for each octave b
Output: $\psi(\vec{x})$

$\psi(\vec{x}) \leftarrow \vec{0}$
for all octave b **do**
    $\psi_x^{(b)} \leftarrow Noise((x,y,z),b)$ using Algorithm 4
    $\psi_y^{(b)} \leftarrow Noise((y,z,x),b)$ using Algorithm 4
    $\psi_z^{(b)} \leftarrow Noise((z,x,y),b)$ using Algorithm 4
    Compute amplitude $A^{(b)}(\vec{x})$ at $\vec{x}$ using $\{E_{ijk}^{(b)}\}$ and Equation 2.6
    $\psi(\vec{x}) \leftarrow \psi(\vec{x}) + A^{(b)}(\vec{x})(\psi_x^{(b)}, \psi_y^{(b)}, \psi_z^{(b)})$

We use a hash function to map each lattice point to the 128-vector-pairs array. Instead of Perlin's hash function, which induces severe axis-aligned artifacts in frequency space (see figure 8(a) and (b) from Cook and DeRose's work [23]) we use the following:

$$\text{hash}(i,j,k) = H(i \text{ xor } H(j \text{ xor } H(k))) \bmod 128 \tag{2.8}$$

where $H(s)$ provides a good quality pseudo-random permutation of the 32-bit integers:

- $H(s)$:

Algorithm 4 Compute Octave Noise at Position $\vec{x}$

Input: position $\vec{x} = (x, y, z)$, octave b, array of basis vectors $basis[128][2]$, grid net rotation $\{\theta_{ijk}^{(b)}\}$
Output: $Noise(\vec{x}, b)$

1: $\delta x^{(b)} \leftarrow 2^{-b+1}\Delta x$
2: Identify grid's cube C_{ijk}: $i \leftarrow \lfloor \frac{x}{\delta x^{(b)}} \rfloor$, $j \leftarrow \lfloor \frac{y}{\delta x^{(b)}} \rfloor$, $k \leftarrow \lfloor \frac{z}{\delta x^{(b)}} \rfloor$
3: Get basis vectors $\{p_l, q_l\}_{l=1}^{8}$ for C_{ijk} nodes using Equation 2.8
4: Get net rotations $\{\theta_l\}_{l=1}^{8}$ for C_{ijk} nodes from $\{\theta_{ijk}^{(b)}\}$
5: Advance gradient vectors $\{\vec{g}_l\}_{l=1}^{8}$ for C_{ijk} nodes using Equations 2.4, 2.5
6: Map gradients $\{\vec{g}_l\}_{l=1}^{8}$ to noise values $\{N_l\}_{l=1}^{8}$ for C_{ijk} nodes
7: Compute noise: $Noise(\vec{x}, b) \leftarrow interpolate(\{N_l\}, \vec{x})$

- $s \leftarrow (s \text{ xor } 2747636419) \cdot 2654435769 \bmod 2^{32}$
- $s \leftarrow (s \text{ xor } \lfloor s/2^{16} \rfloor) \cdot 2654435769 \bmod 2^{32}$
- $s \leftarrow (s \text{ xor } \lfloor s/2^{16} \rfloor) \cdot 2654435769 \bmod 2^{32}$
- return s

This has in fact proven useful as a reasonably efficient stateless pseudo-random number generator in other work.

For rendering we typically run many smoke marker particles through the total velocity field (seeded in the appropriate places for the given simulation), evaluating velocity wherever and whenever needed. With several octaves of noise, all the interpolation and gradient evaluations can be rather more expensive than simple interpolation of velocity from a grid, even with optimizations such as our trigonometic approximation. However, we do highlight that it is embarrasingly parallel—each marker particle can be advected without reference to any of the others—and the amount of data involved is fairly small relative to the apparent resolution of the turbulent velocity field. Further optimization, however, may be possible by using more memory: evaluating at least some of the octaves on higher resolution grids once and then interpolating from there, or evaluating on smaller tiles that are cached for reuse by multiple particles.

A pseudocode for the small-scale simulation is provided in Algorithm 2.

2.6 Temporal Coherence

We inject turbulence energy in the smoke source region; in our implementation we used a fixed initial value. During the large-scale simulation the energy goes through spatial diffusion and spectral cascade processes that preserve temporal coherence. We then derive from the turbulence energies two types of noise quantities: amplitude and net rotation, computed on the lattice points of each octave. These values are computed using continuous function (square root and multiplication) on a temporally coherent energy field, hence form a temporally coherent mapping. Inside grid cells we smoothly interpolate the amplitude and net rotation values in a C^1 continuity manner. The noise function at a point is then computed from the net rotation there, which changes smoothly in space and in time, applied to the auxiliary array that is fixed throughout the simulation; hence it is temporally coherent. The potential function at a point is a sum over octaves of noise and amplitude multiplications, hence do not induce any temporal violation. Finally the small scale velocity field uses the curl of potential values, and as a result the sub-grid particle velocities change smoothly in time.

2.7 Reducing Angular Dissipation in Simulation

We have so far looked at layering in sub-grid turbulence; it is of course also imperative that as much simulation detail as possible is retained in the large-scale simulation. We propose a simple multistep predictor to reduce the aforementioned loss of angular momentum caused by the first order time splitting of advection.

To motivate our method, we point out that the incompressibility constraint can be arrived at as the infinite limit of the second viscosity coefficient λ:

$$\frac{d\vec{u}}{dt} = \frac{\lambda}{\rho}\nabla(\nabla \cdot \vec{u}) + f \tag{2.9}$$

where f in our case is buoyancy forces. (Note this is not the physically correct limit in which compressible flow becomes asymptotically incompressible, but rather a useful thought experiment which avoids the need of considering variables other than velocity and equations other than momentum.) As $\lambda \to \infty$, any divergent motions are damped out to nothing, giving back incompressible flow; other motions,

such as shearing, are unaffected.

For a stiff problem such as equation 2.9, an implicit integrator with stiff decay is recommended. For example, if Backwards Euler is used for the stiff viscosity term, and we then take the limit of the solution as $\lambda \to \infty$, we find we are back to the usual pressure projection splitting method: $\vec{u}_{n+1}$ is the divergence-free part of the advected $\vec{u}_n$.

To get higher accuracy we consider better implicit methods which still posess stiff decay, such as BDF2. This is a multistep method which, after again assuming the advection term is handled separately, would discretize equation 2.9 as:

$$\vec{u}_{n+1} = \frac{4}{3}\vec{u}_n - \frac{1}{3}\vec{u}_{n-1} + \frac{2}{3}\Delta t \frac{\lambda}{\rho}\nabla(\nabla \cdot \vec{u}_{n+1}) + \frac{2}{3}\Delta t f_{n+1} \tag{2.10}$$

If we solve this, then take the limit as $\lambda \to \infty$, we get pressure projection at time $n+1$ of

$$\vec{u}^*_{n+1} = \frac{4}{3}\vec{u}_n - \frac{1}{3}\vec{u}_{n-1} + \frac{2}{3}\Delta t f_{n+1}, \tag{2.11}$$

to get $\vec{u}_{n+1}$, which can be seen as a simple predictor for the new velocity. (We emphasize it is the final time $n+1$ velocity that is made divergence-free: there is no divergence error in the velocity field in which we advect particles.) See figure 2.6 for a graphic illustration of this effect in a rotational flow.

A graph of the kinetic energy sum over grid points in a 2D FLIP simulation is presented in Figure 2.5. The velocity predictor reduces dissipation and hence more energy is measured in all steps throughout the entire run compared to a regular FLIP scheme. We also ran a large scale smoke simulation with and without our velocity predictor. Snapshots are shown in Figure 2.8 and are further discussed in Section 2.8.

A more theoretical analysis of this scheme is left for future work, e.g. quantify the contribution in other scenarios such as rigid body rotation, or investigate the error under this scheme when transferring quantities from particles to the grid and vice versa. However our preliminary experiments with FLIP, where we transfer the predicted new particle velocities, based on their current and past values, to the grid for pressure projection instead of the current velocity, are very promising: the method remains apparently unconditionally stable, with significantly improved

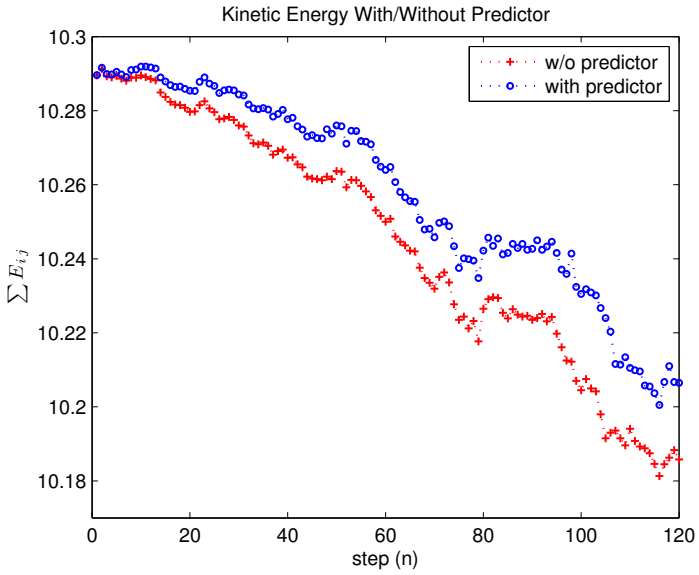


Figure 2.5: Kinetic Energy with Enhanced FLIP. Kinetic energy sum over grid points in a 2D FLIP simulation, running on a unit square without external forces, and initialized with divergence free velocity field. Blue: with velocity predictor (enhanced FLIP). Red: without velocity predictor (regular FLIP). Enhanced FLIP better preserves kinetic energy.

conservation of angular momentum yet without introducing noise. Visually flows remain a lot more lively and areas of rotation are damped much more slowly, without blowing up.

2.8 Results

We ran several demonstrations of the sub-grid turbulence model in 3D. The time added to the simulation for tracking the extra turbulence quantities was negligible, giving roughly two seconds per frame for a $30 \times 120 \times 30$ grid on a 2.2Ghz Core Duo. Running the marker particles through the total velocity field with three octaves of noise scaled linearly with particle count, at about .3 seconds per frame for 1000 particles, adding 260% overhead to the large scale computation time. This became a bottleneck for higher quality renders with hundreds of thousands of particles, but as we noted earlier should allow for trivial parallelization not to mention

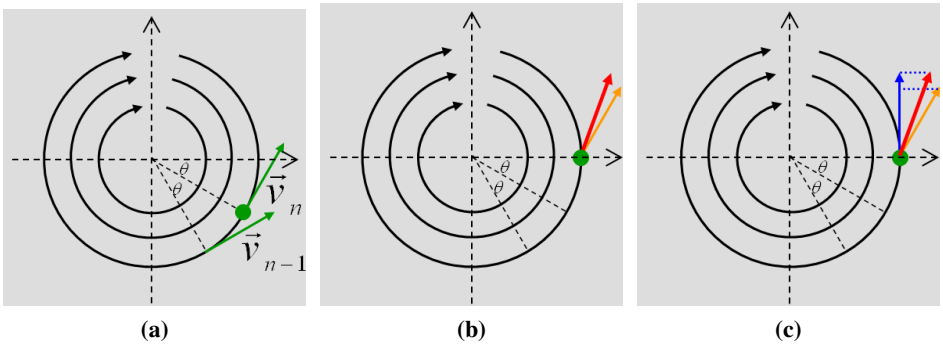


Figure 2.6: Angular Velocity in a Rotational Flow. (a) green: velocity v_n, v_{n-1}. (b) red: advection step $\tilde{v}_{n+1}$ using Equation 2.11, orange: regular advection. (c) blue: projection step v_{n+1} of red velocity better preserves angular momentum.

other optimization possibilities.

Figure 2.2 shows frames from a simulation of a smoke plume generated by a continuous source of hot smoke, with the full turbulence model in place. Marker particles were seeded in each frame, then rendered with self-shadowing.

Figure 2.7 demonstrates the effect of layering the turbulence model on top of existing large-scale motion, with or without the evolution (diffusion and spectral cascade). Without turbulence the motion is flat and boring, at least in these early frames. With turbulence but without evolution, we were forced to include energy in all octaves right from the start; there thus appears to be too much fine-scale detail initially, but not enough mid-scale turbulent mixing. The full evolution model instead shows the spectral cascade, with the smallest-scale vortices only appearing naturally as energy reaches them, and the spatial diffusion lets the turbulent mixing extend further into the flow breaking up symmetries better.

Figure 2.8 shows a side by side series of frames from two large-scale simulations (i.e. without sub-grid turbulence effect). The smoke plume generated with velocity predictor rises faster and shows more vivid motion, presumably connected to lower dissipation. The simulations used grid of size 30x120x30 with 27k FLIP particles, and smoke buoyancy was injected in the beginning of the simulation.

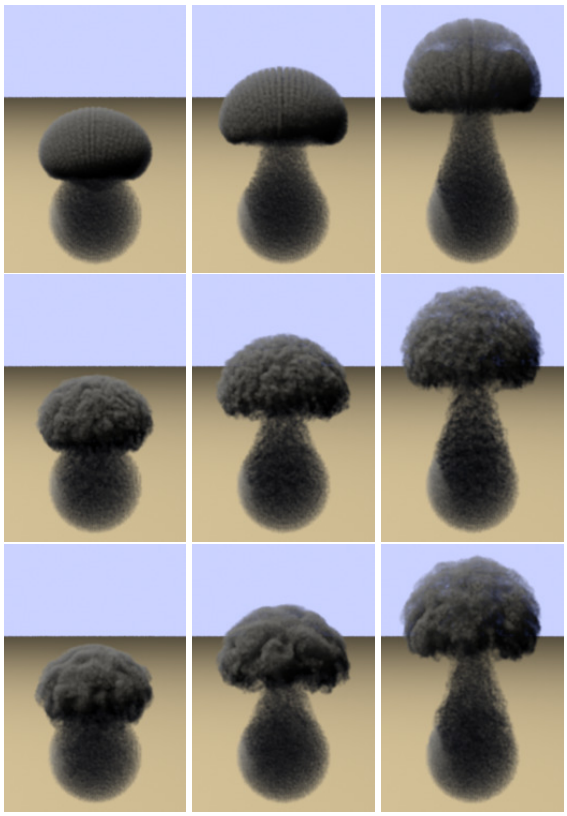


Figure 2.7: Sub-Grid and Energy Transport Effects. Top row: large-scale motion only. Middle row: turbulence added, but without evolution. Bottom row: our full turbulence evolution model.

2.9 Conclusions

We presented both a new sub-grid turbulence evolution model and a new predictor step for fluid simulation, with the goal of getting closer to high quality smoke simulation. At present our model remains too simplistic for scientific validity, but we argue it captures much of the qualitative look of real turbulence.

There are many directions for future work, beyond simply improving the accuracy of our model: in particular, we highlight the question of how to automatically pull energy from the large-scale simulation into the first turbulent octave. Ideally a solution to this would not perturb stable laminar regions of flow, but naturally

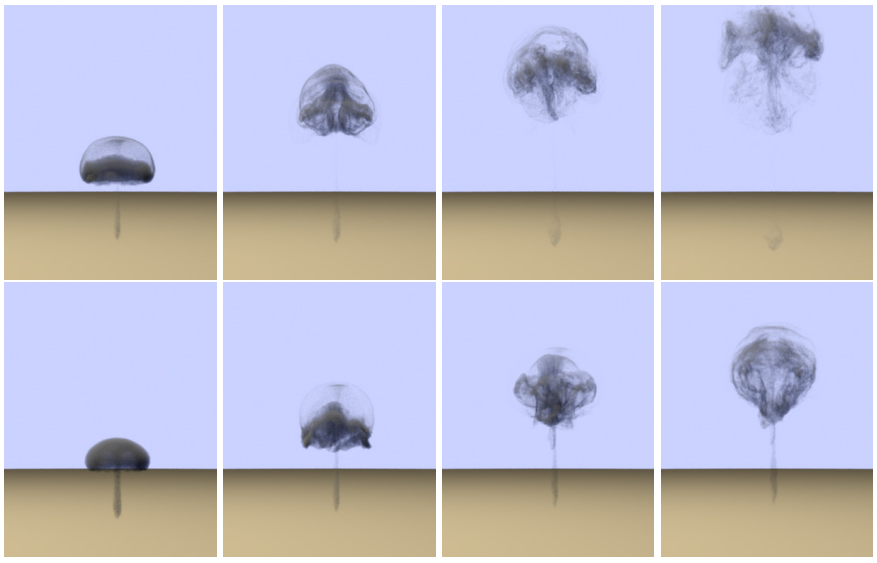


Figure 2.8: Frames With/Without Velocity Predictor. Top row: enhanced FLIP using velocity predictor. Bottom row: regular FLIP without velocity predictor. (equivalent frames are presented.) Smoke rises faster and has livelier motion in the enhanced FLIP scheme, presumably connected to lower dissipation.

cause a transition to turbulence when an instability is detected.

Chapter 3

Ghost SPH for Animating Water

3.1 Introduction

Animating liquids like water via physical simulation remains a beguiling area. Many methods have been developed in graphics to solve various problems, both for realism of the results and for performance. Our focus here is Smoothed Particle Hydrodynamics (SPH), where the fluid is represented by material particles with relatively simple interactions via sums of smooth kernel functions and their gradients: see Monaghan for a detailed overview [48], which we assume as background knowledge. Most SPH methods enjoy automatic mass conservation (each particle represents a fixed amount of conserved mass, density is estimated by direct summation), accurate tracking of the fluid through space (due to the Lagrangian representation), and a conceptually simple algorithmic kernel (summing weighted kernels over nearby particles, with no systems of equations to solve or tricky geometric discretizations).

However, several problems remain: here we tackle the treatment of free surface and solid boundary conditions. Mass-conserving SPH methods, where density is estimated by a sum rather than evolved as a state variable, suffer serious errors near free surfaces causing unavoidable surface-tension-like artifacts and reduced stability. Prior treatments of arbitrary solid boundaries do not reflect the correct cohesive behaviour visible in many real world scenarios. Our Ghost SPH method resolves both by careful sampling of ghost particles in the air and solid regions near

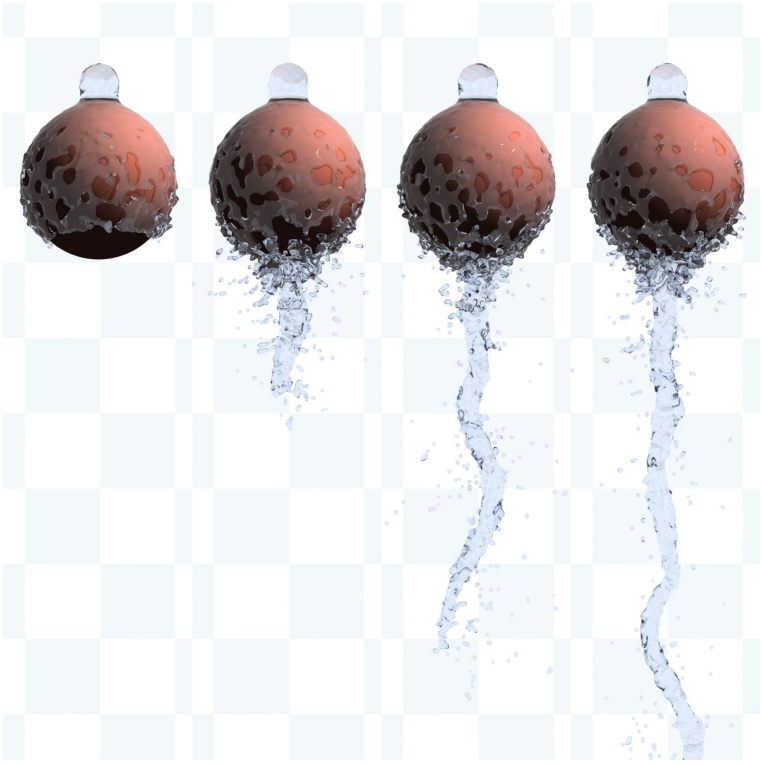


Figure 3.1: Tomato Under Tap Water. With our new boundary conditions, water pouring on to a sphere properly coheres to the underside, giving a realistic stream without excessive numerical surface tension.

the fluid, and appropriate extrapolation of fluid quantities to the ghost particles. Our new sampling algorithm, an extension of a Poisson-disk method to accurately capture boundaries, is also of interest to other applications.

3.2 Related Work

SPH was independently introduced by Gingold and Monaghan [34] and Lucy [45], and has since seen extensive use in physics research. The advantages above have also made it popular in computer graphics for a variety of liquid phenomena.

Monaghan's adaptation [47] of SPH to free-surface flow serves as a basis for many later works. Müller et al. [49] used a low stiffness equation of state along

with surface tension and viscosity forces for interactive applications. Refined particle sampling near free surfaces for accuracy or efficiency is discussed in several works [1, 40, 66]. Becker and Teschner [7] reduce compressibility with the stiffer "Tait Equation", and introduce particle initialization with highly damped equations to reach a stable density near the surface, which we directly solve with ghost air particles.

Müller et al. seeded air particles to model bubbles [49]. Keiser also used a narrow band of air particles to aid in surface tension and small bubbles [40] — however, whereas this work's mirroring approach may create air sampling with widely varying density, we always seed ghost air particles with a near-rest-state distribution.

Bonet and Kulasegaram [13] modified the underlying SPH method with corrections to make the scheme numerically consistent, resolving density problems near boundaries as we do, but with significantly heavier calculation.

Müller et al [50] achieved two-way SPH fluid interaction using "color field"-derived curvature. Solenthaler and Pajarola [67] introduced a modified density equation for accurate SPH density computation at the interface between the two fluids. However, this is not directly applicable to free surfaces.

Many SPH works represent solids with particles. Repulsion forces from solid particles are often used to avoid fluid penetrating solids [47]. Two-way interaction between fluid and solid particles is also common [9, 40]. Colagrossi and Landrini more accurately captured velocity boundary conditions at solids by mirroring nearby fluid particles, but were limited to simple solid geometry [21]. We handle velocities in a similar manner, but with solid bodies of arbitrary shape. Ihmsen et al. [39], like us, extrapolated fluid quantities into solid particles to improve solid boundary conditions, avoiding "stacking" and irregular pressure distributions in standing water situations; we further capture accurate cohesion to solids.

Fedkiw and collaborators' "Ghost Fluid Method" [30], used extensively for Eulerian fluids including coupling to Lagrangian solids [29], inspires our velocity and density extrapolation to ghost particles at both solid and free surface boundaries.

Artificial viscosity is critical in SPH for stability and regularity. Many methods use Lucy's equations [45], e.g. [7, 47]. Müller et al. suggested another formulation based on second derivatives of a specially designed kernel [49, 50]. Our artificial

viscosity approach instead derives from the Monaghan's "XSPH" position update for handling shocks [46]; we extend this to a velocity update, and find it more intuitive and controllable than prior forms.

Solenthaler and Pajarola's PCISPH [68] replaces the equation of state, with its stability time step restriction, by an iterative solver for bringing the system to incompressibility. PCISPH allows much larger time steps and reduces the need for parameter tuning. While our examples use an equation of state, we use the same summed-mass density and thus our Ghost SPH method can just as easily be used with PCISPH.

Our seeding algorithm is based on blue noise sampling original developed for rendering. Cook [22] emphasized the virtues of Poisson-disk distributions for blue noise. Turk [73] introduced a relaxation method for surface sampling. Dunbar et al. [25] modified dart throwing to efficiently generate Poisson-disk distributions in 2D; Bridson [16] simplified this with rejection sampling, extending it to higher dimensions with minimal implementation effort. We extend Bridson's algorithm to tightly sample the boundary of a domain as well as its interior.

3.3 The Basic Method

There is a wide variety of SPH methods; here we introduce our notation and present a common set of choices for graphics, which we call "Basic SPH". We also call attention to the problems we later will solve with our Ghost-SPH model.

Particle i has position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, and mass m_i (uniform across all particles), with acceleration $\mathbf{a}_i$ computed from force divided by m_i. Kernel function W is radially symmetric with a finite radius of support: our examples use Monaghan's M_4 cubic spline [48] with radius of support $3h$ for an average particle spacing of h, but this is orthogonal to the contributions of the paper.

Density Evaluation: One critical choice is how to compute density ρ_i at particle i. We advocate the popular direct summation estimate,

$$\rho_i = \sum_j m_j W_{ij}, \tag{3.1}$$

where $W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j)$. This naturally conserves mass, and forces computed from

this density directly correct deficiencies in the particle distribution, generally producing higher quality results.

Equation (3.1) is problematic near free surfaces. In the interior particles are surrounded by other particles, so an even distribution gives a roughly uniform density. Near a free surface, however, the air part of a particle's neighborhood is empty and the same distribution gives a much lower density estimate: see Figure 3.3. The equation of state then causes particles to unnaturally cluster in a shell around the liquid, rebalancing density but causing a strong surface-tension-like artifact, stability and accuracy issues due to the distorted distribution, and a deformed initial shape as in Figure 3.4b and d.

An alternative is to evolve the density as an independent quantity [47]:

$$\partial\rho/\partial t + \nabla\cdot(\rho\mathbf{v}) = 0.$$

This eliminates the free surface problem, but discretization and integration errors in this equation permit drift from true mass conservation, leading to steadily worse particle distributions particularly around splashes.

Solid Boundaries: We represent solids by sampling with particles. The most common way of preventing liquid particles from penetrating solids is to apply repulsive forces near solid particles: our Basic-SPH examples use the Lennard-Jones approach advanced by Monaghan [47]. However, while this aims to prevent liquid from entering a solid, it freely allows liquid to separate from a solid — which is often just wrong. In typical scenarios, liquid can only separate from a solid if air can rush in to fill the gap: by default liquid shouldn't be able to leave, leading to the standard no-stick $\mathbf{v}\cdot\hat{\mathbf{n}} = 0$ boundary condition. This provides a visually critical form of cohesion, quite apart from surface tension, which our ghost treatment of solid boundaries enables (see Figures 3.1 and 3.12 as well as the accompanying video with real footage).

Solid Collisions: Despite treating boundary conditions at the velocity level, truncation errors in time integration may allow liquid particle positions to stray inside solids. Therefore we check them against the solid geometry, represented by level sets for convenience, and if inside project them back to the outside.

Incompressibility: In our examples we computed pressure from density using

the Tait equation, $p_i = k\left((\rho_i/\rho_0)^7 - 1\right)$ [47] with $k = 2000$, but other equations or calculations such as PCISPH may serve equally well for the purposes of this paper.

Pressure Force: We used Monaghan's usual pressure gradient, $-m_i \sum_j m_j \left[p_j/\rho_j^2 + p_i/\rho_i^2\right] \nabla W_{ij}$ to compute the pressure force on particle i, but other formulas could be freely used.

XSPH: Noise in the raw particle velocities can make the simple position update $\mathbf{x}_i^{\text{new}} = \mathbf{x}_i + \delta t \cdot \mathbf{v}_i$ problematic. XSPH [46] improves matters by blending in surrounding particle velocities with the addition of $\varepsilon \sum_j (m_j/\rho_j)(\mathbf{v}_j - \mathbf{v}_i) W_{ij}$ where ε is a user-tuned parameter on the order of 0.5.

Artificial Viscosity: Some form of artificial viscosity is necessary to stabilize the inviscid equations. For Basic SPH we adopt the same model as Becker and Teschner [7] with $\alpha = 0.0005$.

3.4 The Ghost SPH Method

3.4.1 Algorithm Overview

We solve the particle deficiency at boundaries and eliminate artifacts by (1) dynamically seeding ghost particles in a layer of air around the liquid with a blue noise distribution, (2) extrapolating the right quantities from the liquid to the air and solid ghost particles to enforce the correct boundary conditions, and (3) using the ghost particles appropriately in summations. Figure 3.2 shows the three classes of particles in a 2D simulation and Algorithm 7 gives full pseudocode for a time step. Mass, density, and velocity are assigned to ghost particles in the spirit of Fedkiw's Ghost Fluid Method: if a quantity should be continuous across a boundary, it is left as is in the ghost; if it is decoupled and may jump discontinuously, the fluid's value is instead extrapolated to the ghost.

3.4.2 XSPH Artificial Viscosity

Before describing the ghost treatment of boundary conditions, we introduce a simplification of the basic method which eases implementation. The goal of artificial viscosity is just to damp out nonphysical oscillations that plague the undamped method: a full treatment of physical viscosity, with a nonphysical coefficient tuned

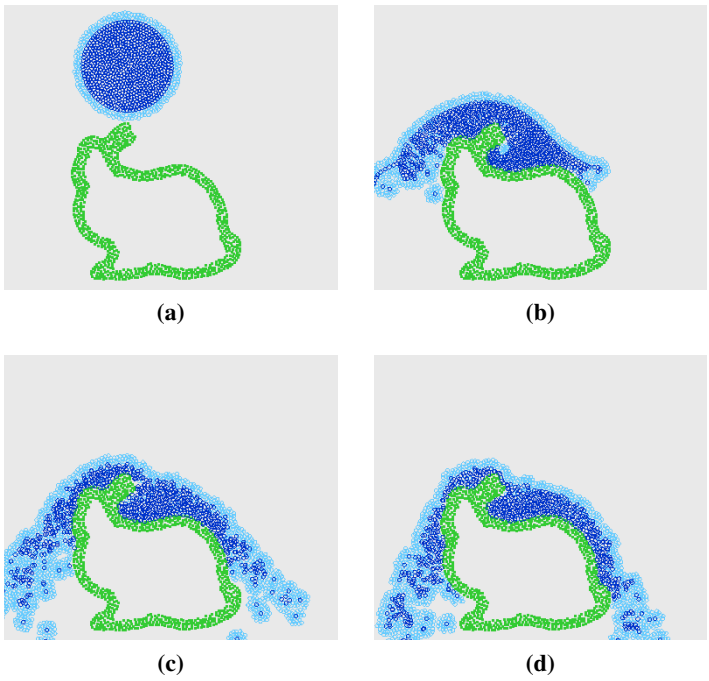


Figure 3.2: Ghost SPH: 2D Simulation Snapshots.
Dark Blue: liquid. Light blue: ghost air. Green: ghost solid.

to stabilize the discretization, is overkill. We propose adopting the simpler XSPH style of damping noise, but at the velocity update level — this works just as well, but is cheaper and easier to tune, and further obviates the need for XSPH in the position update.

Every time step we take the step-advanced velocities $\mathbf{v}_i^* = \mathbf{v}_i + \delta t \cdot \mathbf{a}_i$ and diffuse them with a tunable parameter $\varepsilon = 0.05$:

$$\mathbf{v}_i^{\text{new}} = \mathbf{v}_i^* + \varepsilon \sum_j \frac{m_j}{\rho_j} \left(\mathbf{v}_j^* - \mathbf{v}_i^*\right) W_{ij}. \tag{3.2}$$

We can then evolve positions directly with the particle velocity, $\mathbf{x}_i^{\text{new}} = \mathbf{x}_i + \delta t\, \mathbf{v}_i^{\text{new}}$.

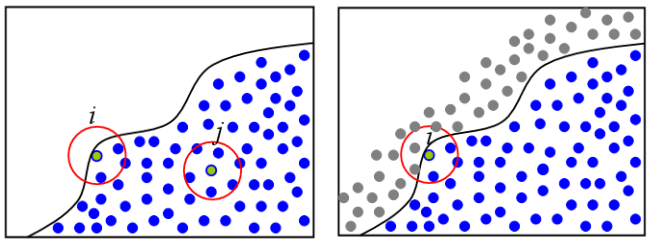


Figure 3.3: Ghost-Air Particles at the Free-Surface. Fluid particles are shown in blue, ghost-air particles are shown in gray. (a) Basic SPH: particle j in the fluid interior has complete neighborhood but particle i at the free surface has partial neighborhood. (b) Ghost SPH: using ghost particles the neighborhood of particle i is now complete.

3.4.3 Ghost Particles in the Air

Air particles are generated at the start of simulation within a kernel radius of the liquid, outside of solids: see Section 3.4.5. The ghost mass of each air particle is set to the mass of a liquid particle, and the ghost velocity of an air particle is set to the velocity of the nearest liquid particle (extrapolating, since in a free surface simulation, there isn't even a defined air mass or velocity). Under the $p = 0$ free surface boundary condition, we set the ghost air density equal to the rest density of the liquid, producing zero pressure.

Ghost air particles contribute to the density summation, and since they fill a thick enough layer around the liquid, this entirely solves the free surface density problem: see Figure 3.3. As their density is the rest density, they contribute no pressure force on the liquid. We explicitly exclude them from the artificial viscosity, as they would add a slight bias in favour of the outermost particles' velocities.

To make the overhead of resampling negligible, we only resample every 10 time steps or so (twice per frame in our examples), and in interim steps advect the ghost air particles with their velocities. This is frequent enough that no appreciable drift of the ghost particles away from the liquid happens in practice.

Figure 3.4 demonstrates the results of a zero-gravity hydrostatic test with Basic SPH vs. Ghost SPH. Our method keeps the fluid stable, maintains its original shape

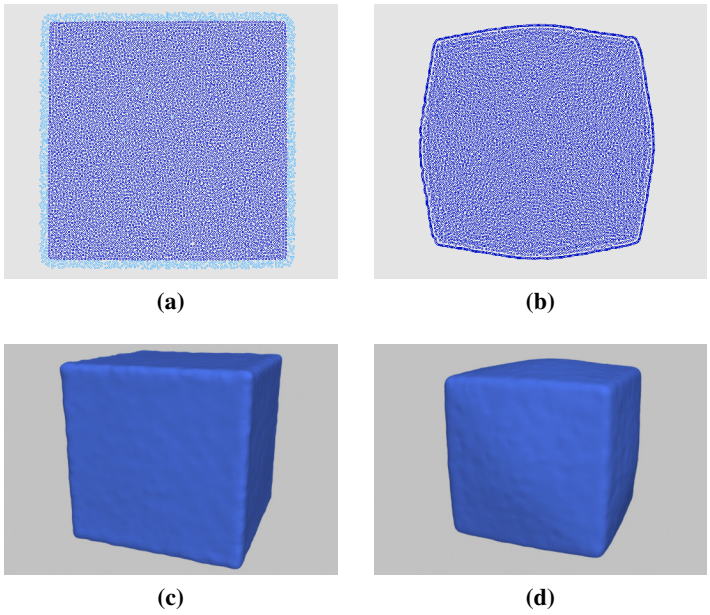


Figure 3.4: Hydrostatic Test. Upper row: 2D square. Lower row: 3D cube. Left: Ghost SPH. Right: Basic SPH. Taken at frame 400.

and volume, and conserves the initial uniform particle distribution and density. In contrast, in the Basic SPH model pressure pushes the outer particles inwards to reach a density equilibrium, forms a stiff shell of particles at the free-surface, and non-uniformly shrinks the fluid volume.

3.4.4 Ghost Particles in the Solid

Solid particles, like air particles, have a dual role: correcting the density summation near the boundary and implementing the right boundary condition in lieu of the basic method's Lennard-Jones approach. They too are seeded inside each solid within a kernel-radius layer of the solid surface, are assigned a ghost mass equal to the liquid particles, and contribute to density summation in the liquid.

The solid boundary condition implies different treatment of ghost density. Pressure should be continuous through the boundary, thus we extrapolate fluid density (which controls pressure via the equation of state) by setting each ghost density to

the nearest liquid particle's density. This naturally enforces the no-penetration *and* no-separation boundary condition (up to numerical errors). Note that while this is physically correct, when the boundary layer is under-resolved, as may happen in large-scale simulations, reverting to freely separating boundary conditions by disallowing negative pressures may provide more plausible results [6, 20].

For an inviscid liquid, the associated no-stick condition implies the normal component of liquid and solid velocities match at the boundary, but that tangential components are completely decoupled: the liquid can freely slip past tangentially. We thus construct the ghost velocity at each solid particle by summing the normal component of the solid's true velocity and the tangential component of the nearest liquid particle's velocity:

$$\mathbf{v}^{\text{ghost}} = \mathbf{v}_N^{\text{solid}} + \mathbf{v}_T^{\text{liquid}}. \tag{3.3}$$

See Figure 3.5. Ghost solid velocities contribute to XSPH / artificial viscosity of Equation 3.2, reinforcing the velocity part of the solid boundary condition without introducing unphysical tangential drag.

Of course, for a visibly viscous liquid, the no-slip boundary condition may be more appropriate, where tangential components of solid and liquid velocities also match. In this case, setting the full ghost solid velocity to the real solid velocity, allowing its tangential component to enter the XSPH artificial viscosity term, gives rise to the desired stickiness; the ε parameter controls the stickiness and may be adjusted independently for the solid for artistic control.

3.4.5 Sampling Algorithm

For seeding particles in the initial liquid, solid layer, and dynamic air layer we need fast isotropic uniform sampling which tightly fits given boundaries. The boundary fit rules out simple periodic patterns: Figure 3.7 illustrates the artifacts caused by simple grid sampling in the air. We turned to Poisson disk patterns, and specifically Bridson's fast rejection-based approach as the simplest to implement and modify in 3D [16]. Throughout we take a parameter r proportional to the desired SPH inter-particle spacing and use it as the Poisson disk radius.

There are several components to our sampling, used depending on the context

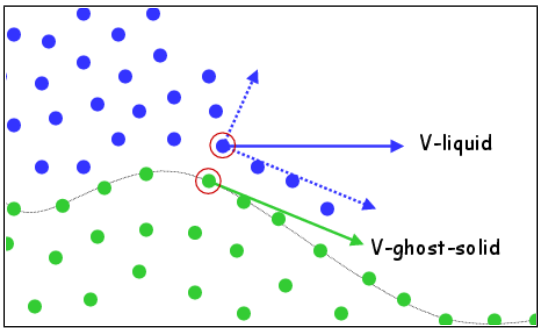


Figure 3.5: Ghost-Solid Velocity in inviscid flow with static solid wall. Blue: liquid particles. Green: ghost-solid particles.

(liquid, solid, or air): sampling on the surface, relaxation on the surface, sampling in the volume, and relaxation in the volume.

For the initial liquid particles, we first sample the surface (represented by a level set for convenience), then improve that initial sampling with surface relaxation, then sample the interior volume, and finish with volume relaxation.

Solids are sampled the same way, but with a distance limit on the volume sampling since we only need a narrow band of particles.

Air sampling, and continuous liquid emission during the simulation, eschew the surface sampling and the relaxation, and instead just sample the required volume starting from existing nearby liquid particles which serve the role of the "surface".

We use similar hashed acceleration grids to expedite the the acceptance/rejection search for point samples as we use for SPH summations. For air sampling we also include liquid and solid particles in the rejection test, as we must avoid sampling too close to the liquid or solid, and need not sample beyond one kernel-radius band of the liquid particles.

While reading through the following details of each step, refer to Figure 3.6 for an illustration of the different components in 2D, and Figure 3.8 for the interior relaxation in particular. Note that the method applies equally to any dimension, and may be applicable to many problems outside SPH.

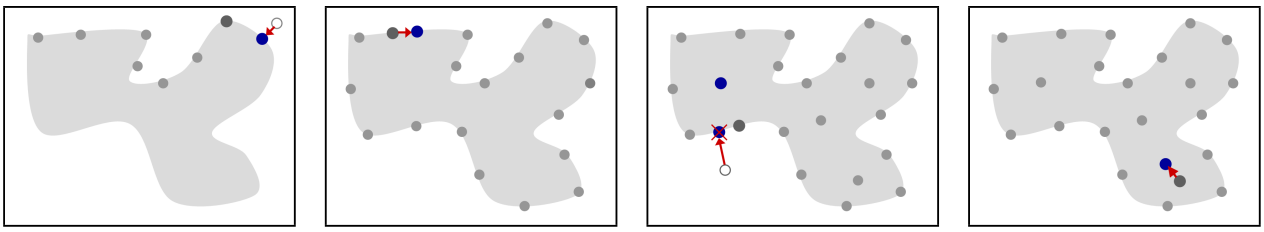


Figure 3.6: Sampling Algorithm Illustration. (a) Sampling the surface. (b) Surface relaxation. (c) Sampling of the interior. (d) Volume relaxation. Blue: newly added or moved particle. Darker gray: particle originating the new particle sample. White with gray line: point sampled to the exterior before projected to the surface.

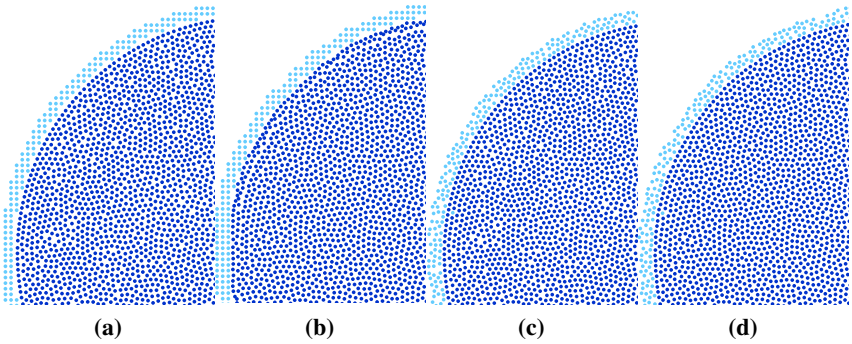


Figure 3.7: Grid Sampling the Air vs. Poisson Disk. (a) Initial grid. (b) 500 frames later, with an anisotropic shell developed. (c) Initial Poisson disk. (d) 500 frames later, essentially unchanged.

Surface Sampling

We assume the surface geometry is given as a signed distance function: this permits fast projection of points to the surface. Pseudocode is provided in Algorithm 5. In the outer loop we search for "seed" sample points on the surface, checking every grid cell that intersects the surface (i.e. where the level set changes sign) so we don't miss any components: in a cell we take up to t attempts, projecting random points from the cell to the surface and stopping when one satisfies the Poisson disk criterion, i.e. is at least distance r from existing samples. Once we have a seed

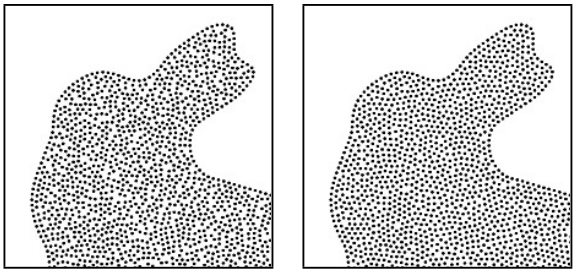


Figure 3.8: Sampling Relaxation Step in 2D. Our sampling algorithm running in a 2D cross section of the Stanford Bunny geometry with zoom in to the head. Left: intermediate result before the last relaxation step. Right: the final result after relaxation.

sample, we continue sampling from it, taking a step of size $e \cdot r$ from the previous sample along a random tangential direction $\mathbf{d}$, again projecting to the surface and checking the Poisson disk criterion. Parameters $t = 30$ and $e = 1.085$ worked well, but could be further tuned.

Interior Sampling

In the interior sampling stage we run regular Fast Poisson Disk Sampling [16] but starting with the surface sample points as initial seeds, and using the level set to reject any samples outside the geometry. For solids, we also use the level set to avoid sampling beyond one kernel-radius into the interior. For air or continuous liquid emission, we use the existing liquid particles as the initial "surface" seeds, and for air also avoid sampling more than a kernel radius away from the liquid. As speed is critical for air and liquid emission during simulation, we reduce the maximum number of random attempts per sample t to 8; for initial liquid shapes and solids we take the usual $t = 30$.

Surface and Volume Relaxation

The goal of the relaxation procedure (Algorithm 6) is to reduce noise in the SPH density by optimizing sample locations. It runs twice during the initial particles

Algorithm 5 Surface Sampling

Input: Level set ϕ, radius r, # attempts t, extension e
Output: Sample set S

for all grid cells C where ϕ changes sign **do**
  for t attempts **do**
    Generate random point $\mathbf{p}$ in C
    Project $\mathbf{p}$ to surface of ϕ
    if $\mathbf{p}$ meets the Poisson Disk criterion in S **then**
      $S \leftarrow S \cup \{\mathbf{p}\}$
      Break
  if no point was found in C **then**
    Continue
  while new samples are found **do**
    Generate random tangential direction $\mathbf{d}$ to surface at $\mathbf{p}$
    $\mathbf{q} \leftarrow \mathbf{p} + \mathbf{d} \cdot e \cdot r$
    Project $\mathbf{q}$ to surface of ϕ
    if $\mathbf{q}$ meets the Poisson Disk criterion in S **then**
      $S \leftarrow S \cup \{\mathbf{q}\}$
      $\mathbf{p} \leftarrow \mathbf{q}$

seeding process, first for relaxing the surface samples and then for relaxing the volume samples.

We attempt to reposition each sample in turn so that it is a greater distance away from its closest neighbor, again using simple rejection sampling. The code takes $t = 50$ nearby random points (with distance from the sample decreasing through the iteration) and if any are further from other samples than the original position, we take the best. Surface sample candidates are additionally projected to the surface of the level set, and interior sample candidates are projected if outside the level set. We sweep through all the samples k times, with $k = 5$ for the surface and $k = 30$ for the volume.

Air Sampling and Fluid Emitter Sampling

We sample air using the "Fast Poisson Disk Sampling" algorithm, starting with the current step fluid particles as an initial sample set. A new point candidate is added to the air sample set if it obeys two rules:

Algorithm 6 Surface/Volume Relaxation

Input: Initial sample set S, level set ϕ, radius r, # sweeps k, and # attempts t
Output: S relaxed sample set

for k sweeps **do**
  for all $\mathbf{p} \in S$ **do**
    Let $B(\mathbf{p}) \subseteq S$ be the samples within $2r$ of $\mathbf{p}$
    $d \leftarrow \min_{\mathbf{q} \in B(\mathbf{p})} \|\mathbf{p} - \mathbf{q}\|$
    $\mathbf{p}^{new} \leftarrow \mathbf{p}$
    for $i = 0 \ldots (t-1)$ **do**
      $\tau \leftarrow \frac{t-i}{t}$
      Generate random vector $\mathbf{f}$ from unit sphere
      $\mathbf{p}^{\text{cand}} \leftarrow \mathbf{p} + r \cdot \tau \cdot \mathbf{f}$
      if $\mathbf{p}^{\text{cand}}$ outside ϕ or came from surface sample **then**
        Project $\mathbf{p}^{\text{cand}}$ to surface of ϕ
      $d' \leftarrow \min_{\mathbf{q} \in B(\mathbf{p})} \|\mathbf{p}^{\text{cand}} - \mathbf{q}\|$
      if $d' > d$ **then**
        $\mathbf{p}^{\text{new}} \leftarrow \mathbf{p}^{\text{cand}}$
        $d \leftarrow d'$
    $\mathbf{p} \leftarrow \mathbf{p}^{\text{new}}$

1. It meets the Poisson Disk criterion that is checked against both fluid particles and air particles in the accumulated sample.

2. It is within kernel radius distance from at least one fluid particle.

The fluid particles hence serve the role of the "surface": a candidate point that is too far away gets rejected. As a simple optimization, we do not sample air particles around isolated liquid particles, as their motion is just ballistic anyhow.

Fluid emitter sampling is done similarly; instead of checking proximity to fluid particles we test if the new candidate point is inside the fluid source geometry.

3.4.6 SPH Density Distribution

Our stochastic sampling doesn't optimally pack samples together. Therefore we use an empirically determined radius of $r = 0.92h$ where h is the desired simulation particle spacing, thereby reaching an SPH density close to the target rest

density. At the initial state, we further measure the average density $\bar{\rho}$ and scale the initial particle mass by $\rho_0/\bar{\rho}$ where ρ_0 is the target rest density: the densities then average exactly to ρ_0, and we start very close to a correct physical equilibrium. In contrast, Basic SPH suffers from noticeable acoustic waves in the beginning of the simulation, which requires several hundred damped steps to reach an initial equilibrium.

Algorithm 7 Ghost SPH Simulation Step

Sample new liquid particles, or air particles, if needed this step.
Compute density:
for all liquid particles i **do**
    Compute ρ_i with Equation 3.1
for all solid particles i **do**
    Find closest liquid particle j
    $\rho_i \leftarrow \rho_j$
Compute pressure:
for all particles i **do**
    Compute pressure p_i using the equation of state
Compute liquid accelerations and velocities:
for all liquid particles i **do**
    Compute the acceleration $\mathbf{a}_i$ from gravity and pressure
    $\mathbf{v}_i^* \leftarrow \mathbf{v}_i + \delta t \cdot \mathbf{a}_i$
Prepare solid boundary conditions:
for all solid particles i **do**
    Find closest liquid particle j
    $\mathbf{v}_i^* \leftarrow \mathbf{v}_i^N + \mathbf{v}_j^T$ as in Equation 3.3
Apply XSPH artificial viscosity:
for all liquid particles i **do**
    Update $\mathbf{v}_i^{\text{new}}$ using Equation 3.2
Extrapolate velocity into air:
for all air particles i **do**
    Find closest liquid particle j
    $\mathbf{v}_i^{\text{new}} \leftarrow \mathbf{v}_j$
Update positions:
for all liquid and air particles i **do**
    $\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \delta t \cdot \mathbf{v}_i^{\text{new}}$

3.4.7 Temporal Coherence

Whenever we resample the air region, there can be a small change in the nearby density summations etc. which in principle could be a problem for temporal coherence. Since we resample twice per frame, this is not an issue — and even if we resampled less frequently we could always smoothly fade out the old air particles while fading in the new. That said, it's worth evaluating how small the change due to resampling is.

Consider a simple constant shear flow where only internal forces (pressure, artificial viscosity) act on the fluid. The exact solution has zero acceleration, which we use as a baseline to measure the general SPH error vs. the change due to air resampling. We ran this example with 6400 liquid particles initially in the unit square centered at $(0,0)$ with velocity field $\mathbf{v}(x,y) = (0,x)$.

We measure the acceleration $\mathbf{A}_i$ of a particle with the second order finite difference of position, and from that define two metrics,

$$E_i^n = \|\mathbf{A}_i(t^n)\| \tag{3.4}$$

$$\Delta E_i^n = \|\mathbf{A}_i(t^n) - \mathbf{A}_i(t^{n-1})\|, \tag{3.5}$$

where t^n is the time at step n. The general SPH error is given by E_i^n, how far the acceleration deviates from the theoretical solution (zero), or in other words the SPH error. We estimate the jump due to resampling by comparing ΔE_i^n on the steps with resampling versus the other steps.

Figure 3.9 shows both the average and the maximum of the metrics over all particles for the first few dozen steps. Every tenth step, after air resampling, we can see a telltale jump in the ΔE_i value. However, the jump is of the same magnitude as the SPH acceleration error E_i in steps without resampling, which is quite tolerable. The accompanying video supports this, demonstrating stable flow in free fall stages and hydrostatic tests for example.

3.5 Results and Discussion

We ran our simulations on a quad-core Intel i7-2600 (8M Cache, 3.40 GHz) with 8GB memory. Neighbor searches and sampling were accelerated with background grid structures. We used OpenMP to parallelize particle computations.

Figures 3.11 and 3.12 show selected frames from our 3D simulation results, referred herein as the "tomato" and the "bunny" examples. Table 3.1 gives statistics on particle counts, overhead for ghost particles, and run time. The tomato example has a continuous emission of liquid particles, thus we report average liquid particle count and computation time over 16k time steps / 800 frames.

Figure 3.12 of the tomato example compares Ghost SPH to Basic SPH with the same shared parameters. Basic SPH suffers severe surface tension artifacts at air and solid interfaces, lacks the physical cohesion between liquid and solid, and causes particles to form unnatural clusters instead of spray. Ghost SPH simulation comes much closer to the natural motion of the fluid flow, wrapping all the way around the tomato before freely leaving in a stream from the bottom — despite the small number of simulation particles. Figure 3.13 shows three real footage series of a tomato under stream of milk and water, emphasising the similarity of the Ghost SPH animation to the liquid flow and cohesion to solid in real-life.

Figure 3.10 of a double dam break simulation compares Ghost SPH to Basic SPH on a larger 750k particle run. Ghost SPH averaged $22.1s$ per step, a computation time that is proportional to the particle count, compared to the Ghost SPH Tomato and Bunny examples of smaller particle counts. Basic SPH averaged $21.5s$ per step, indicating that the dominant cost factor in this setup was fluid/solid interaction, hence the Ghost SPH overhead per step was marginal. The Basic SPH simulation suffers from stiff solid boundary forces, which in turn demand a much smaller time step for stability: 10^{-5} in Basic SPH compared to 10^{-4} in Ghost SPH. Hence on equivalent simulation steps a preliminary scene is reached in the Basic SPH setup compared to the Ghost SPH. Consequently Ghost SPH benefits from a major performance improvement compared to Basic SPH.

The ghost air particles of Ghost SPH added a nontrivial 141% memory overhead in the tomato example, though a much smaller 74% overhead for the bunny. Asymptotically the memory overhead scales with the surface area, not the volume,

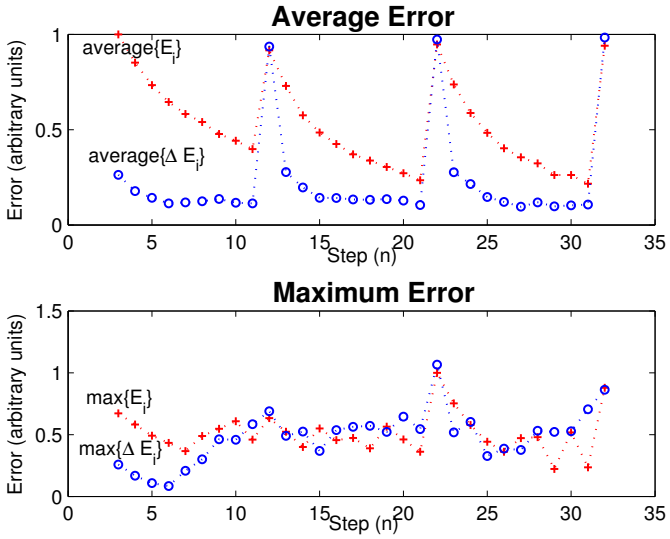
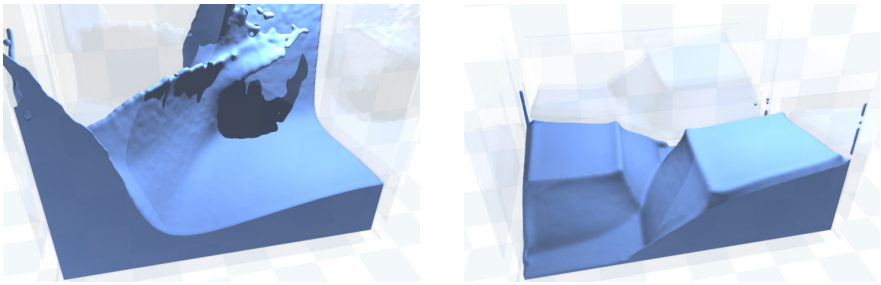


Figure 3.9: SPH Error vs. Resampling Error. Measurements of the general SPH error E_i (red) and the acceleration change ΔE_i (blue) in simulation steps $3-32$. Top: average values. Bottom: maximum values. Each graph is normalized by its maximum red plot value. Resampling occurs at every tenth step.

and thus the overhead is reduced at higher resolutions. Interestingly, Ghost SPH only took 26% more computation time per time step than Basic SPH for the tomato, $1.29s$ versus $1.02s$: we argue the considerable improvement in results at the same resolution is worth this small cost. The computation overhead cause is the air resampling step and the increased number of particles; on average air resampling took 11% of a simulation step computation time, density and particle neighbor data 56%, pressure force 19%, and XSPH for artificial viscosity and boundary conditions took 10% of the time. We also found the improved particle distribution at boundaries meant a much wider range of stiffness coefficients and artificial viscosities could work, simplifying parameter tuning; in some but not all simulations a much larger stable time step was possible than with Basic SPH, which lead to a net improvement in performance.



(a) (b)

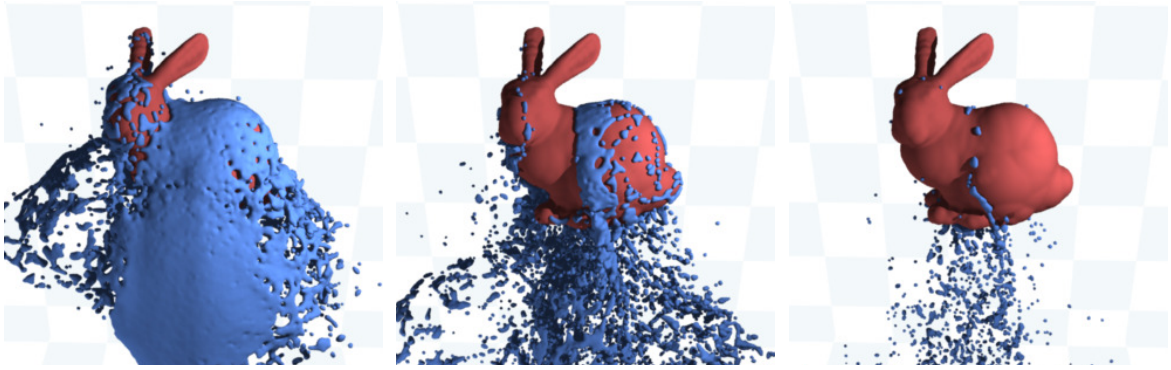
Figure 3.10: Double Dam Break. A larger 750k particle simulation of a double dam break, side by side comparison at frame 250. Left: Ghost SPH. Right: Basic SPH. Stiff solid boundary forces in the Basic SPH simulation require reducing the time step for stability; as a result at equivalent steps Basic SPH reached a preliminary scene compared to the Ghost SPH simulation.

3.6 Conclusions

Together our ghost treatment of solid and free surface boundaries, particle sampling algorithms, and new XSPH artificial viscosity allow significantly higher quality liquid simulations than basic SPH with only a moderate overhead. Our approach should extend easily to related methods such as PCISPH, and the sampling may find use in any particle simulation. Looking to the future, we plan to extend the ghost method to a more accurate treatment of surface tension and to two-way coupling with solids or other fluids.

Statistic	Tomato	Bunny
Liquid particle average count (#)	72k	88k
Ghost-solid particle overhead	187%	109%
Ghost-air particle overhead	141%	74%
Avg. computation time per step	1.29s	0.88s
Avg. computation time per frame	25.8s	17.6s

Table 3.1: Simulation statistics for the 3D examples. "Overhead" refers to the ratio of ghost particles to real liquid particles. Each animation frame is subdivided into 20 time steps.



(a) **(b)** **(c)**

Figure 3.11: Fluid Sphere on Solid Bunny.

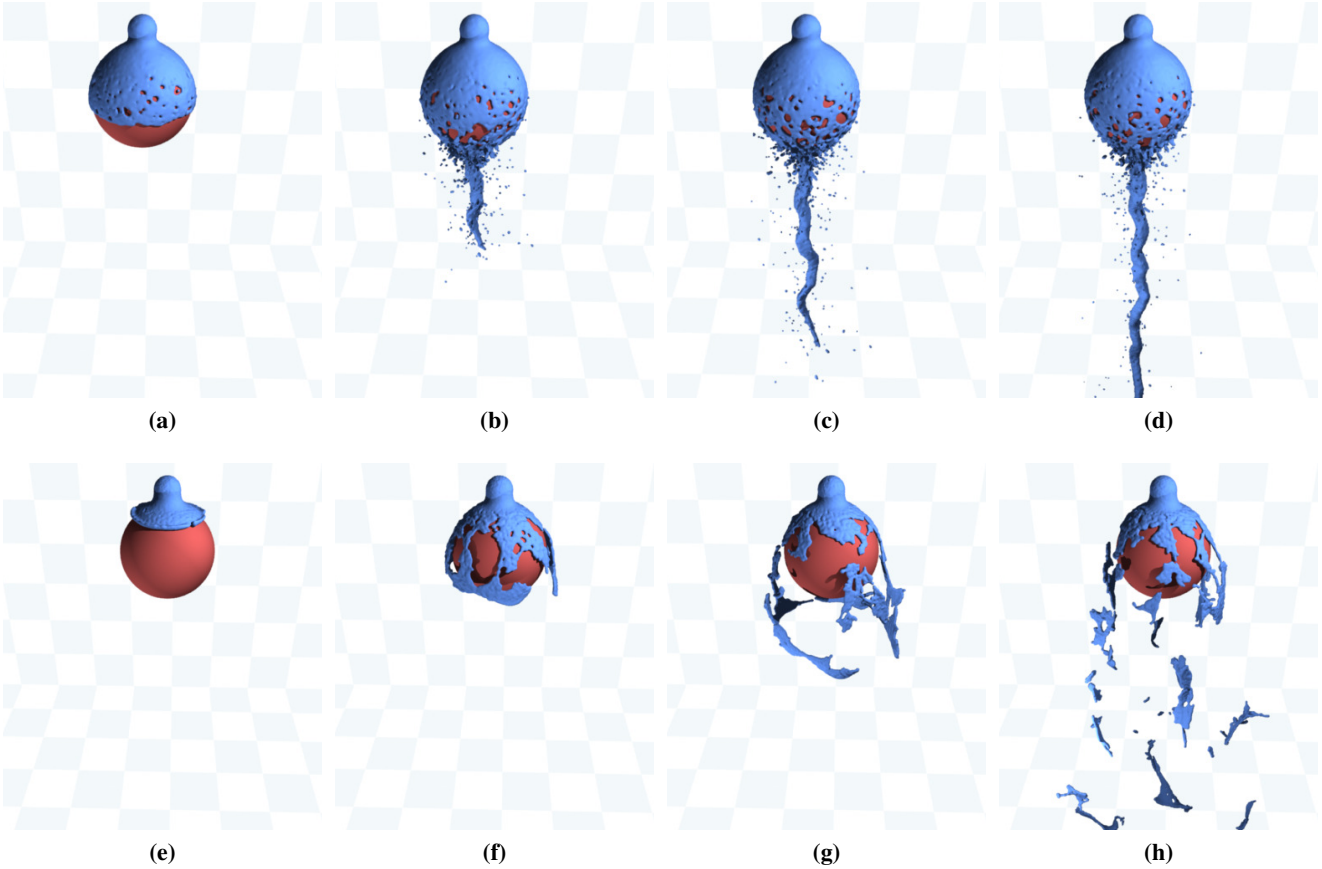


Figure 3.12: Tomato Under Tap Water. Ghost SPH vs. Basic SPH. Upper row (a)-(d): Ghost SPH. Lower row (e)-(h): Basic SPH. (equivalent frames are presented)

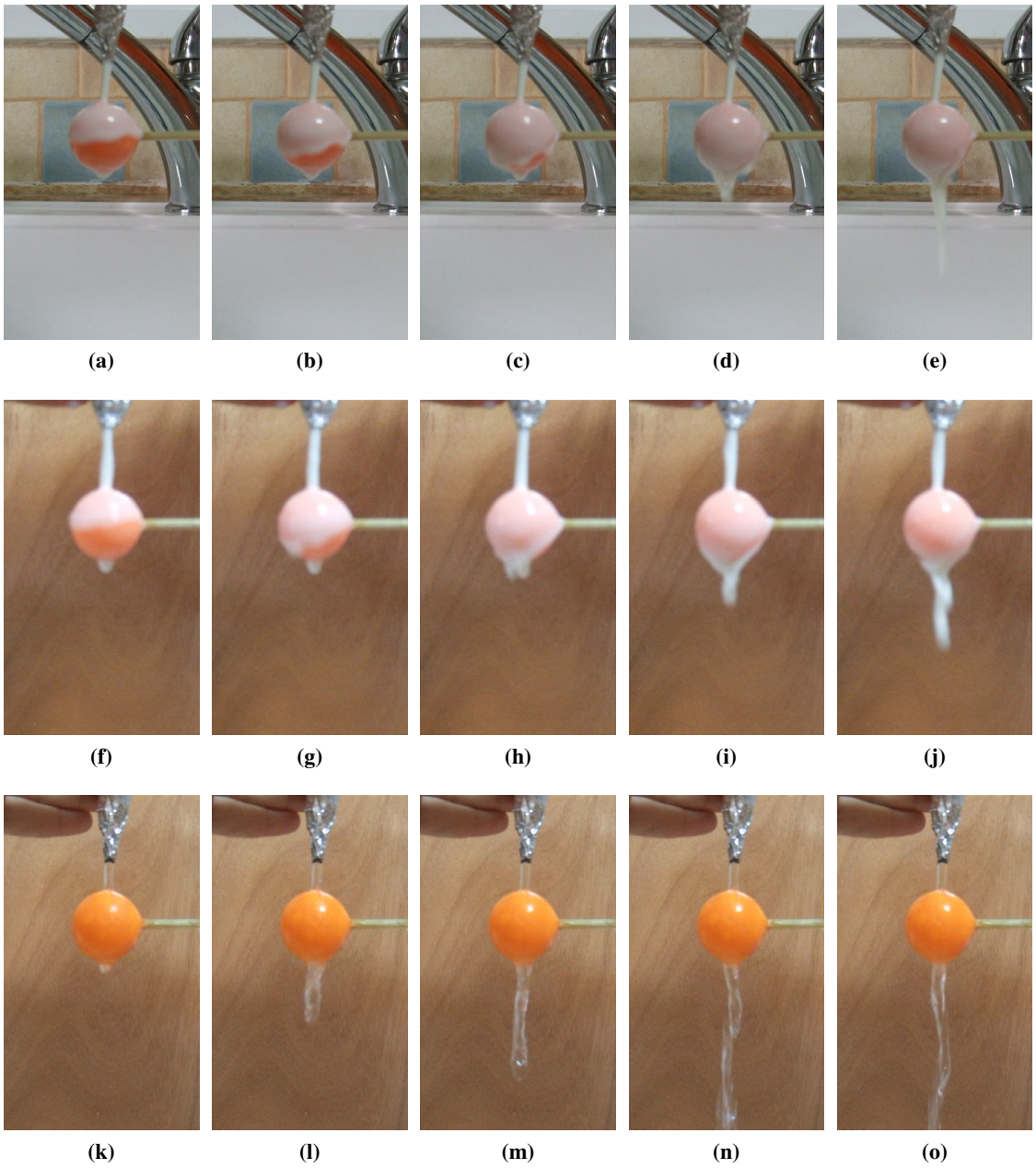


Figure 3.13: Tomato Under Liquid Stream. Real Footage Frames. Consecutive time frames (left to right) of tomato under stream of milk (top, middle) and water (bottom), showing cohesion of liquids to solids in real-life.

Chapter 4

The Beta Mesh: a New Approach for Temporally Coherent Particle Skinning

4.1 Introduction

Fully three-dimensional liquid animation using physics-based simulation is now common in feature film, and is becoming attractive for interactive applications. In this chapter we focus not on the underlying dynamics, discussed earlier in this thesis, but rather on the geometric problem of tracking, capturing, or reconstructing the surface of the liquid throughout a free-surface liquid simulation.

In this chapter, we present exploratory work in the area of surface reconstruction from particle fluids, both identifying an issue (temporal coherence) which has yet to be thoroughly addressed, and proposing new algorithms which bring us closer to a solution. We assembled a convincing algorithm for 2D, and made progress in extending it to 3D, while at the same time realizing new ideas will still be required to fully address the challenges in 3D. In particular, we mean by "temporal coherence" that we want the geometry to change continuously in time w.r.t. the motion of the underlying particles, even if the topology of its tessellation may discretely change arbitrarily. The challenge we directly address is producing

continuously varying geometry despite the underlying topology used in the construction algorithm changing abruptly in time.

Two classes of surfacing methods dominate the field of fully three-dimensional[1] free-surface liquid animation: implicit Eulerian methods and particle-based mesh-free methods. In the latter, also known as Lagrangian methods, the liquid volume is sampled by particles which are advected in space by the governing equations. From the beginnings of liquid animation, these methods have been popular due to various advantages such as inherent mass conservation and ease of implementation.

Particles fill the liquid volume roughly uniformly, generally with some level of irregular sampling depending on the underlying simulation algorithm. The particles move by physical laws and hence follow smooth trajectories (up to discretization limitations), can form varying topologies through splitting and merging, and evolve into smooth surfaces as well as fine features of various kinds.

SPH is a popular Lagrangian approach successfully used in Computer Graphics to simulate a wide range of phenomena and applications, such as free surface flow [49], two-phase flow [50, 67], bubbles [37], porous flow [44], and even viscoplastic solids [8, 24]. FLIP has also been a popular choice in recent years for simulating different phenomena such as water and sand [77], smoke [60], and multi-phase flow [14]. Particle-based methods such as these do not guarantee homogenous distribution among particles. In typical "weakly compressible" SPH the stiffness of the Equation of State relating pressure to density determines the level of compressibility, which is reflected in sampling variations, as well as small-scale irregularities inherent in the kernel-based smoothing approach and sometimes artifacts such as clustering. FLIP enforces the fluid velocity incompressibility on the scale of a background grid, but lacks a built-in mechanism to control how particles are spread in the domain at a sub-grid resolution; FLIP is also robust to quite irregular distributions which is sometimes exploited deliberately by increasing particle sampling in a thin shell next to the surface. Moreover, in both of these techniques, once the liquid starts splashing and forming thin features, or interacts with solid walls, the particles are likely to be unevenly distributed in the domain.

[1] We exclude heightfield methods, very commonly used for ocean surfaces among other things, as being 2.5 dimensional; here displacements applied to an underlying surface mesh already work extremely well.

The last step in such a simulation pipeline is rendering the liquid surface reconstructed from particles to form a realistic animation. Creating a surface representation from the particles is necessary to use standard rendering techniques. Moreover, representing the surface as a mesh, that presumably forms a tessellation of a smooth limit surface, is more amenable to further geometric processing, rendering with displacement textures, specular reflection and refraction, motion blur, etc.

The crucial surface characteristics we aim for can be classified in two groups. The first family of properties is related to the fluid geometry at an instant in time. We desire smooth liquid surfaces (particularly apparent in specular reflection or refraction), as well as the maintenance of interesting details, such as ripples, droplets, tendrils, and thin sheets. These properties have been the focus of sustained research throughout the years, and is relatively well-explored even if still under-appreciated in the academic graphics research community. The second family of properties relates to the temporal coherence of the animated surface. We want the surface to evolve smoothly in time while performing natural fluid merges and splits, and avoiding any pops or jitters that reflect surfacing limitations. Opposed to the spatial surface properties, the temporal property has had very little attention in related research, and is the primary focus of our work.

We here propose a surfacing technique dubbed the "beta mesh", for skinning particles in a temporally coherent manner, while attempting to capture smooth surfaces and fine features. At present it does not quite achieve all goals, but we believe it introduces some techniques worthy of further study and shows promise in many common scenarios.

Two prior surfacing techniques which inspired our work are essential building blocks. The *union of balls* introduced in the early stages of computer graphics research, is totally faithful to particle positions, and hence evolves smoothly in time. However only in an infinitesimal limit it can capture the flatness of the surface, and in a standard simulation flow the surface will have spherical bumps. The *alpha shape* [27, 28] is a dual form of the union of balls [26], can be directly converted to a mesh form, the *alpha mesh*, by keeping vertices and connectivity, and replacing arcs with edges. The alpha mesh can reproduce flat surfaces despite irregularity in particle sampling, and in fact no other method known to us can do it similarly.

On the other hand, the alpha mesh lacks temporal coherence as we will explain later: moving a set of vertices in the mesh by an arbitrarily small amount can cause an edge flip in a non-planar region, thin features can connect and disconnect from one step to the next, and new vertices can appear or disappear from the mesh in a nonsmooth manner.

The beta mesh algorithm attempts to benefit from the best properties of these two techniques. While we define it in terms of the union of balls, and thus inherit many of the good properties of the union of balls, our algorithm for construction is based on the alpha mesh, making use of duality. We also include a temporally-coherent smoothing post-processing operation which admits a temporally-coherent and spatially smooth limit surface, potentially of use to other constructions.

Our beta mesh technique is not fully resolved yet. While in 2D it clearly demonstrates the desired properties, our 3D implementation still requires further research in order to meet our goals. In 3D the beta mesh has a lot more topological problems. The bigger challenge is the definition of beta faces: while it is unambiguous in 2D and moreover provides a unique transversal on the alpha mesh vertices, in 3D ambiguity is present, especially on thin sheets, surface holes, and their stitching to the fluid volumetric parts. The construction can reveal beta faces that are not connected at all, or are not connected in straightforward way, which makes the mesh design very challenging. This also leads to difficulties in post-processing tasks such as expanding the beta mesh volume to include all the alpha mesh vertices. In addition, thin features such as tendrils, represented in the alpha mesh as degenerate edges (i.e. with no triangles), do not have a representation in the beta mesh in its current form.

4.2 Related Work

Since the introduction of particle systems, direct particle rendering has always been a popular choice for certain applications [59, 64]. In the context of liquids, this is appropriate for spray or foam, but for smoother, more coherent liquid surfaces — especially with reflection and refraction effects — this is a nontrivial undertaking.

The *union of balls* algorithm wraps each particle in a sphere of a given radius. Given smooth particle trajectories, the outer surface of the union of balls also

changes smoothly in time as it is entirely faithful to particle positions. However by construction it cannot represent flat surfaces other than in the theoretical limit of infinitesimal particle spacing, and with nonzero spacing the surface suffers from kinks (discontinuous normals) at the intersection between overlapping spheres.

To the best of our knowledge the union of balls representation has not been directly used for rendering, however it is often used as a guide in various contexts, e.g. cell fraction computation in MAC grids for accurate pressure solves [5], clipping a Voronoi diagram of the particles [65], and determining the signed distance field in a one phase flow [32] and a two phase flow [14].

The foundational algorithm for smooth surface reconstruction from particles was introduced by Blinn [12]. A summation of Gaussian density distribution functions is used, resulting in a surface that is often called "metaballs", or "blobbies", since it generalizes balls. It too is temporally coherent, but while the surface is smooth (in fact, with Gaussians, it is infinitely differentiable) it has a hard time simultaneously approximating flat surfaces and thin features. Typically there is no middle ground between bumpy "cottage cheese"-like reconstruction and overly "blobby" but smooth reconstruction which loses features. Blinn's approach has been the basis for various later works which attempt to overcome this difficulty.

The *alpha shape* was first introduced in 2D [28], and later on was extended to 3D [27]. Its associated mesh captures flat surfaces exceptionally well, and provides an extremely sharp and detailed reconstruction of tendrils and thin features. However, it does not transition in time smoothly, which is critical for surfacing simulation data. Edelsbrunner et al. [26] demonstrated its properties as the dual form of the union of balls, which is of particular interest for us, raising the question of how the good properties of the union of balls and the alpha mesh can be merged in a single algorithmic framework.

Solving for an implicit levelset function without re-initialization has been used in [57] to reconstruct the surface. Avoiding levelset re-initialization helps to get smooth transitions, however may deviate in time from the real surface due to numerical errors. Moreover the surfacing computation time in this method was significantly more than the actual simulation time, which is undesirable, and constituted a time-dependent problem in itself ruling out the possibility of parallel reconstruction across all frames independently.

Several works in the past decade focused on improving the surface smoothness in direct surface reconstruction from particles. Müeller et al. [49] defined the surface as an isosurface of a scalar field that is smoothed using an isotropic kernel. Zhu and Bridson [77] used an implicit function of distance to a sphere, replacing the actual position and radius with a computed weighted sum position of nearby particle centers, and a weighted sum radius of particle radii. The distance function is then sampled on a grid, a smoothing pass over the grid is performed, and an isosurface is then extracted from the grid. Their results are considerably smoother than the classic blobby spheres surface when extracting detailed features. Adams et al. [1] refined the algorithm by tracking the particle-to-surface distances over time, and using it in the weighted distance to sphere equation. They experimented with their technique on adaptively-sized particle radii, as well as on homogenous radii. Further improvement was introduced by Yu and Turk [76]. They detect the anisotropy direction of particle positions by performing a Principal Component Analysis over the covariance matrix of particle neighborhood. Then they perform a smoothing step that re-positions the centers of these smoothing kernels. Compensating for the anisotropy in the flow significantly reduces the surface bumpiness. The above direct surfacing from particles improve the reconstructed surface smoothness, however none of them enforces or handles the temporal coherence requirement.

An alternative approach for surface reconstruction was proposed by Williams [74]. A nonlinear optimization problem is solved iteratively to achieve optimal smoothness for the surface mesh, while constrained to lie between two union of balls surfaces of different radii centered at the particles. Bhatacharya et al. [11] introduced a variant of this technique using level sets rather than meshes. Perfectly flat surfaces can be generated using these methods under certain conditions, however some drawbacks are present. First, small changes in particle positions may lead to quite large surface changes since the optimization is global and only geared towards surface flatness. As a result there is no guarantee on reaching smooth transitions of the surface in time.

Another group of works deals with surface reconstruction from a point cloud, e.g. Alexa et al. [2]. We believe that these works face different challenges than ours. While they deal with denoising point positions in order to construct proper

geometry, our objective is in some sense opposite. In our case the particle positions are the ground truth, and our goal is to stick to it while constructing a surface surrounding them in the best way and conforming to certain requirements. On the other hand, some tools and techniques are common to the two classes of surfacing problems. We would like to highlight three works from this group that are of a particular relevance to our work. Bernardini et al. [10] build a manifold subset of the alpha shape similar to our alpha mesh construction, but they do it incrementally. Amenta et al. [3] approximate the medial axis and its inverse and then compute the surface that lies between them. The power diagram used in their construction is closely related to the alpha mesh that we use. Tam and Heidrich [71] compute the dual shape of the weighted Delaunay triangulation, extract singular points, and triangulate guided by the dual shape triangles. The Delaunay dual shape is in fact the alpha mesh, which drives the triangulation of both their technique and ours. Nevertheless several major differences are present. First, our main purpose is processing data originating from an actual particle simulation, and addressing the temporal coherence requirement. This poses new challenges in our case, particularly in the interpretation of thin sheets of different kinds, and dealing with unique geometrical layouts of fine simulation features that are not likely to be present in a given well defined geometry. We cannot fill in gaps in the particle data to compensate for poor distribution sections, and our algorithm needs to handle such segments robustly. In addition we aim producing a smooth limit surface that approximates the liquid's natural smoothness, and yet providing a proper geometrical representation for degenerate alpha simplices.

4.3 The Algorithm

4.3.1 Overview

We assume we are in d-dimensional Euclidean space: points can be drawn from $\mathbb{R}^d$, and the distance between points is the usual Euclidean distance. We are most interested in $d = 3$, but the case of $d = 2$ is also instructive.

A *closed ball* of radius r centered on a point p is the set of points within distance r of p: $\{x : \|x - p\| \leq r\}$.

The *open ball* is where the distance is strictly less than r: $\{x : \|x-p\| < r\}$.

The open ball is the *interior* of the closed ball.

The points at distance exactly r from p form the *boundary* of the ball (whether open or closed).

For the following we will assume a set of n special points $\{x_i\}_{i=1}^n$ which we will call *particles*. For our application these will in fact be fluid particles from a simulation.

An *empty ball* is an open ball with no particles in its interior.

The *union of balls* of the particles is the union of n closed balls centered on each of the particles. It defines a surface around the particles that is temporally coherent, but cannot characterize flat surfaces. The balls have the same radius r, which is fixed ahead of time and is correlated with the particles typical spacing dx.

The algorithm proceeds in three steps: the alpha mesh construction described in Section 4.3.2, the alpha graph processing described in Section 4.3.3, and the beta mesh construction described in Section 4.3.4. Each of these sections includes an overview/definitions subsection that provides mathematical definitions and algorithm overview, and a detailed algorithm subsection that discusses the practical implementation in 3D, provides 2D/3D illustrations, and outlines a pseudocode of the algorithm or parts of it.

4.3.2 Alpha Mesh

Alpha Mesh - Overview and Definitions

The *alpha shape* for a set of vertices $S = \{x_i\}_{i=1}^n$ is a subset of vertices and arcs connecting them, s.t. every arc is part of a closed ball that contains d vertices of S and its open ball form is an empty ball. It may be easier to think of it as the complement of the union of all empty balls.

We say a particle is an *alpha vertex* if it appears on the boundary of the alpha shape.

We will from now on assume the alpha vertices are in "general position". In particular, we will assume:

- no two alpha vertices are coincident,
- no more than d may lie on the boundary of an empty ball where d is the dimension of the space, and
- for each set of d alpha vertices, if there is any ball whose boundary contains them then the center of the ball cannot be on the (hyper-)plane through the alpha vertices.

The first and last assumptions imply that if d alpha vertices appear on the boundary of a ball, they appear on exactly two such balls — one on each side of the plane through the alpha vertices, corresponding to the two possible normal vectors for the plane. Using some variation of the right-hand-rule, we can similarly associate each of the two balls with one of the two orientations of the d alpha vertices.

The *alpha mesh* is an oriented mesh formed on the alpha vertices, with simplex facets (triangles in 3D) corresponding to the ball-shaped sections on the boundary of the alpha shape. More precisely, an oriented facet of d alpha vertices appears in the alpha mesh if they appear on the boundary of an empty ball, and in fact the unique ball associated with the oriented facet as above is empty. The normal of the face points towards the center of that empty ball, or in other words is an outward-pointing normal. See Figure 4.1 for an illustration of the alpha shape and the alpha mesh, or [28] for more illustrations.

Note that there may be *degenerate* simplices in the alpha mesh that are not part of any facet: maximal collections of $d-1$ or less vertices which appear on the boundary of an empty ball. These are not naturally oriented, or can be thought of as appearing with all possible orientations. An *isolated vertex* is not connected to any other vertex. In particular, every closed ball containing the vertex is empty of other particles. An *isolated edge* in the alpha mesh is a pair of alpha vertices distance r or less apart for which every ball containing them is empty of all other particles.

Alpha Mesh - Algorithm Details

Our algorithm runs through all the particles in the set. For for every particle and for every ordered pair in the set within specified distance from the particle, it examines

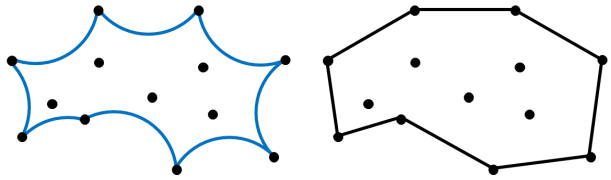


Figure 4.1: Alpha Shape and Alpha Mesh. Particles are shown in black dots. (a) Alpha shape (blue). (b) Alpha mesh (back): alpha shape arcs were converted to edges.

the circumscribing sphere of the ordered triplet. If no other particle in the set is contained in the sphere it declares the ordered triplet as a triangle in the mesh. A pseudocode for the alpha mesh construction in 3D is given in Algorithm 8, and the circumscribing sphere criterion is described in Algorithm 9.

Algorithm 8 Alpha mesh construction

Input: set of particle positions $P = \{\mathbf{p}_i\}$, radius r
Output: set of triangles $T = \{t_i\}$

for all $\mathbf{p}_i \in P$ **do**
  let $N_i = \{\mathbf{p}'_i\}$ s.t. $\mathbf{p}'_i \neq \mathbf{p}_i$, $dist(\mathbf{p}'_i, \mathbf{p}_i) <= 2r$
  for all pair $\mathbf{p}_j, \mathbf{p}_k \in N_i$ s.t. $\mathbf{p}_j \neq \mathbf{p}_k$ **do**
    $(succeeded, \mathbf{c}) = build\_sphere(< \mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k >, r)$
    if succeeded **then**
      if $dist(\mathbf{p}, \mathbf{c}) > r \; \forall \mathbf{p} \in N_i \setminus \{\mathbf{p}_j, \mathbf{p}_k\}$ **then**
        $t =< \mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k >$
        $T \leftarrow T \cup \{t\}$

Our algorithm does not rely on a Delaunay triangulation preprocess like Edelsbrunner's original alpha mesh algorithm [27, 28]. This leads to several advantages:

- Our algorithm runs in $O(n)$ computation time where n is the number of particles, assuming reasonably well-distributed particles. The overhead of the full Delaunay construction is avoided, which takes $O(n\log(n))$ or even worse with 3D thin tendrils particle data.

- Our method is trivial to parallelize: any triple can be tested independently of

Algorithm 9 Build circumscribing sphere

Input: triplet $< \mathbf{x0}, \mathbf{x1}, \mathbf{x2} >$, radius r
Output: flag *succeeded*, center $\mathbf{c}$

$\mathbf{n} = (\mathbf{x0} - \mathbf{x1}) \times (\mathbf{x1} - \mathbf{x2})$
$radius_x = \frac{dist(\mathbf{x0},\mathbf{x1})dist(\mathbf{x1},\mathbf{x2})dist(\mathbf{x2},\mathbf{x0})}{2\|(\mathbf{n})\|}$
if $radius_x > r$ **then**
    return **false**
else
    $\alpha = \frac{dist(\mathbf{x1},\mathbf{x2})^2((\mathbf{x0}-\mathbf{x1})\cdot(\mathbf{x0}-\mathbf{x2}))}{2\|\mathbf{n}\|^2}$
    $\beta = \frac{dist(\mathbf{x0},\mathbf{x2})^2((\mathbf{x1}-\mathbf{x0})\cdot(\mathbf{x1}-\mathbf{x2}))}{2\|\mathbf{n}\|^2}$
    $\gamma = \frac{dist(\mathbf{x0},\mathbf{x1})^2((\mathbf{x2}-\mathbf{x0})\cdot(\mathbf{x2}-\mathbf{x1}))}{2\|\mathbf{n}\|^2}$
    $\mathbf{l} = \alpha\mathbf{x0} + \beta\mathbf{x1} + \gamma\mathbf{x2}$
    $t = \sqrt{(radius_x{}^2 - r^2)/\|\mathbf{n}\|^2}$
    $\mathbf{c} = \mathbf{l} + t\mathbf{n}$
    return **true**

the other triples.

- We find the degenerate parts of the alpha mesh that are missed in the obvious Delaunay construction, and hence can directly address them.

The choice of the search radius has a critical impact on the resulting alpha mesh, as was illustrated in Edelsbrunner's work [27]. A value that is too small may fail connecting relevant geometry parts, a value that is too big may wrongly inflate concave surface areas. Here it also influences the alpha graph and beta mesh derived from the alpha mesh, see the following Section 4.3.3 and Section 4.3.4. In our experiments we used $r = 1.5dx$ where dx is the typical particle spacing, however more tests are required to reach best utilization of our mesh construction.

The local nature of our algorithm however causes sensitivity to rounding errors that may result in an invalid mesh, which is eliminated in the original Delaunay-driven alpha mesh construction. An illustrating example is 4 particles that form a square of dx edge length. All or none of the triangles may be added to the alpha mesh, instead of adding precisely two triangles that share a diagonal. Another example is a very dense cluster of particles: our construction may miss a dense triplet triangulation, causing an opening in the cluster's outer surface. To avoid

such problems we validate the alpha mesh using a simple edge count test: every oriented edge in the mesh must have a pair edge in an opposite direction. If the check fails we perturb the particle positions by a small tolerance and rerun the alpha mesh construction. On the vast majority of the frames that we processed a single alpha mesh pass was used, and only a few required re-running the algorithm once more.

We call attention to several special cases in the alpha mesh structure. A thin layer of fluid that forms a fairly flat sheet of particles, namely a *thin sheet*, is a common scenario that causes degeneracy in the mesh. The alpha mesh representation of a thin sheet in 3D is a double-sided sheet of triangles, clamping the fluid particles on both sides. Each oriented triangle in the thin sheet has an associated triangle with the opposite orientation: this pair forms a *double-sided triangle*. We highlight that the alpha mesh representation of thin sheets is unique, due to the thin and fairly flat surface that is formed. In contrast the union of balls technique produces a bumpy surface in this case. Double-sided triangles may be present in other scenarios as well, such as when shared between two tangent fluid bubbles. The 3D *degenerate simplices* that are not included in the alpha mesh are isolated particles and isolated edges; currently we collect those particles into a set of external particles and render them as spheres.

4.3.3 Alpha Graph

Alpha Graph - Overview and Definitions

We can impose a manifold-like adjacency structure on the d-simplex faces of the alpha mesh. Consider two faces with $d-1$ alpha vertices in common, and look at all faces incident on that $d-1$ simplex: they may be sorted radially by angle around the common $d-1$ simplex. The two faces are adjacent if no other of the co-incident faces appears radially between them, where between is defined as the outward radial section or between where their normals point.

We structure the alpha graph information in *graph faces*, each composed of an alpha vertex and an ordered set of d-simplices. The graph faces do not intersect each other by the construction described above, other than when two simplex

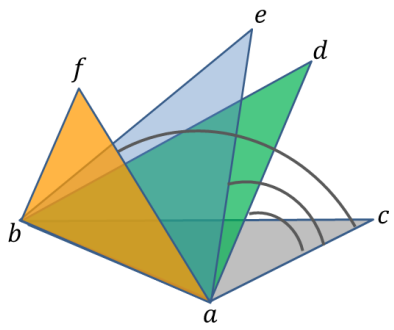


Figure 4.2: Triangles Order in Alpha Face Construction. The order between triangles that share an edge is determined by the signed volume of the tetrahedra, or equivalently by the dihedral angle between two triangles. This sorting guarantees that faces do not intersect.

boundaries overlap. An alpha vertex can impose more than one graph face.

While the non-degenerate parts of the alpha mesh (the d-simplex faces) form a closed and oriented sub-mesh, it might well not be manifold. More than two faces may be incident on a common $d-1$ simplex.

Note that although the above definitions may stand for $d > 3$, handling the graph connectivity in high dimensions is quite complicated; we focus here on our primary interest where $2 \leq d \leq 3$ and neglect higher dimensions.

Alpha Graph - Algorithm Details

The algorithm starts by assigning for each alpha vertex v_i its adjacency set, i.e. the alpha triangles that it participates in. We then divide those triangles into faces: disjoint groups of ordered triangle. We define an order among the vertex triangles: t_j follows t_i w.r.t. v_i if they share an edge in opposite direction that includes v_i. Double-sided triangles make an exception: a triangle cannot follow its other side pair triangle. This definition is sufficient to impose a unique order when there is only one triangle that follows each triangle in the adjacency set, however often this is not the case. When two triangles t_j and t_k follow triangle t_i, we want to connect them by the dihedral angle they form with t_i. We can avoid calculating the dihedral

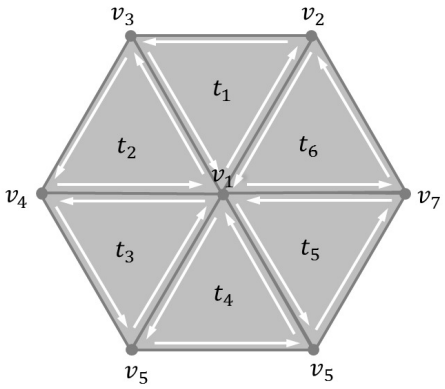


Figure 4.3: Alpha Graph Example. An illustration of the alpha graph of a thin sheet. Alpha mesh is presented in gray triangles/dots, triangle orientation is highlighted in white. v_1 participates in two faces: the front face and the back face, both consist of six triangles. The front face around v_1 is the order set $\{t_1, t_2, t_3, t_4, t_5, t_6\}$; it is a closed face. The faces around vertices v_2-v_7 are open faces. v_2 for example participates in two faces: one on each side, each consists of two triangles, the front face is the ordered set $\{t_1, t_6\}$.

angles and instead use a robust signed volume predicate: if the signed volume of the tetrahedron (a),(b),(d),(e) in Figure 4.2 is positive it implies the dihedral angle to the triangle with (e) is larger; if negative the dihedral angle to the triangle with (d) is larger. This comparison operator is then all the sorting algorithm needs to find the correct dihedral angle order. A *closed face* is when the first triangle in the ordered set follows the last triangle. On the rim of a thin sheet each *boundary face* forms an *open face*, i.e. the last triangle in the ordered set does not have a next triangle. Note that thin sheet boundary faces in 2D are processed seamlessly: the opposite side edge can be considered as a possible next edge, and if no other edge takes precedence it means that the thin sheet boundary has reached and traversal continues on the opposite side. In contrast, the 3D boundary faces on both sides are treated as open faces and the graph construction algorithm identifies the start/end triangles when processing the face. See Figure 4.3 for an illustration of a thin sheet and its alpha faces. A pseudocode for the alpha graph algorithm is provided

in Algorithm 10.

Algorithm 10 Alpha graph

Input: alpha vertices $P = \{p_i\}$, alpha triangles $T = \{t_i\}$
Output: $\forall p \in P$, faces of p $\{f_p^k\}_k$

for all $p \in P$ **do**
  let $T_p = \{t_i\}$ be the set of triangles that include p
  for all $t_i \in T_p$ **do**
    if t_i is processed **continue**
    let $T_p^i \subseteq T_p$ be the set of triangles that follow t_i
    sort T_p^i by signed volume w.r.t. t_i
    $t_{curr} \leftarrow t_j$, $t_{first} \leftarrow t_j$
    while true **do**
      mark t_{curr} as processed
      find the first available triangle $t_j \in T_p^i$
      declare: t_j follows t_{curr}
      if t_{first} follows t_j **then**
        declare: face f complete
        break
      $t_{curr} \leftarrow t_j$
      $T_p^i \leftarrow T_p^i \setminus \{t_j\}$

An illustrative example for an alpha vertex that participates in more than one face is when a fluid bubble is tangent to the fluid surface. If the touching area between the bubble and the air surface include a single vertex, every triangle in the vertex adjacency list has a single triangle that follows it without requiring for dihedral angle sorting. If the touching area is composed of more than one vertex, the dihedral angle sorting determines how to arrange the triangles in faces. In both cases every shared vertex has two faces, one for the air surface and the other for the bubble surface.

4.3.4 Beta Mesh

Beta Mesh - Overview and Definitions

The beta mesh is loosely the dual of the alpha mesh of a set of points.

A *pure beta vertex* can be defined in two equivalent ways. In d dimensions, it

is any intersection of the boundaries of d different balls centered on the particles, which is not inside any open ball centered on a particle — think of it as a point on the boundary of the union-of-balls where d balls meet. Such a point is equivalently the center of a ball whose boundary contains d particles and whose interior is empty of particles. From this it is clear a beta vertex is "dual" to a facet from the alpha mesh. We use the preface "pure" for this type of vertices to distinguish it from other types of beta vertices that will later be defined.

The *pure beta mesh* is an oriented mesh formed on the pure beta vertices, with general faces corresponding to the connected ball-shaped parts of the union-of-balls boundary. These faces may have an arbitrary number of sides, and are often not planar — think of them more as a topological construction rather than well-defined geometry.

For each oriented d-simplex in the alpha mesh, a pure beta vertex can be constructed as the center of the unique corresponding empty ball. Each edge of the beta mesh corresponds precisely to a pair of adjacent alpha mesh faces with no intervening alpha mesh faces between them (going angularly around the common subsimplex). Given the alpha graph structure described above, the beta mesh can be built quite easily: there is a one-to-one correspondence between graph faces and beta faces. Alpha vertices and beta faces are also tightly coupled, yet it is not a one-to-one relationship. Multiple beta faces can share the same alpha vertex – when the alpha vertex has more than one graph face. In this case there is a one-to-many correspondence between alpha vertices and beta faces. At the same time, degenerate parts of the alpha mesh such as isolated particles or degenerate edges have no associated beta vertices or faces.

In 2D the pure beta mesh is a well defined mesh. However in 3D it is not a feasible geometrical representation, and we aim turning it into a triangular mesh, to easily visualize the surface as well as apply advanced rendering tasks such as motion blur. To turn the pure beta mesh into a triangular mesh, we define a new type of beta vertex, the *alphabeta vertex*, each corresponds to a graph face in a one-to-one relationship. We initially position the alphabeta vertices at the projection of the alpha vertex to the approximated beta face plane, and triangulate them with the pure beta vertices of the face.

The initial positions of beta vertices however does not yield the desired smooth

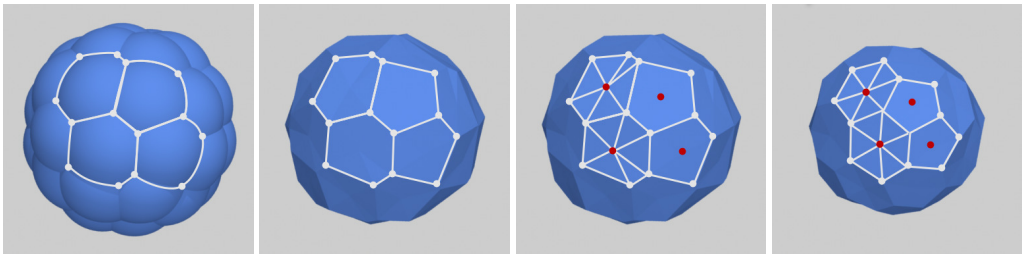
surface. Beta vertex positions are prone to particle distribution fluctuations, reflected in slight alpha triangulation variation that get amplified when converted into beta positions. The alphabeta vertices positioned on the faces planes do not respect the curvature of curved surfaces. Aiming at a smooth surface we hence run a smoothing step. First, we smooth each pure beta vertex by computing a weighted average of all the pure beta vertices in adjacent faces: the center point of every face edge is weighted by the edge length. After updating all the pure beta vertices, we similarly reposition each alphabeta vertex using a weighted average of all the smoothed pure beta vertices in its face. Note that boundary beta vertices do not go through the smoothing procedure to prevent them from getting pushed to the thin sheet interior.

The beta mesh may have degeneracies related to degeneracies in the alpha mesh — or rather, missing features. An isolated vertex in the alpha mesh corresponds to a ball in the union-of-balls which doesn't intersect any other ball, and thus has no associated beta mesh vertices: it vanishes from the beta mesh. In general an isolated $d-1$ or less simplex in the alpha mesh does not contain any proper beta mesh vertices since they are formed from the intersection of d balls, and thus these features are missing from the beta mesh. In order to faithfully capture the motion of the liquid and present splashes and drops that separate and merge into the fluid geometry, we define a third type of beta vertex: the *external beta vertex*. These vertices do not take part in the beta mesh, and instead are rendered as spheres.

Beta Mesh - Algorithm Details

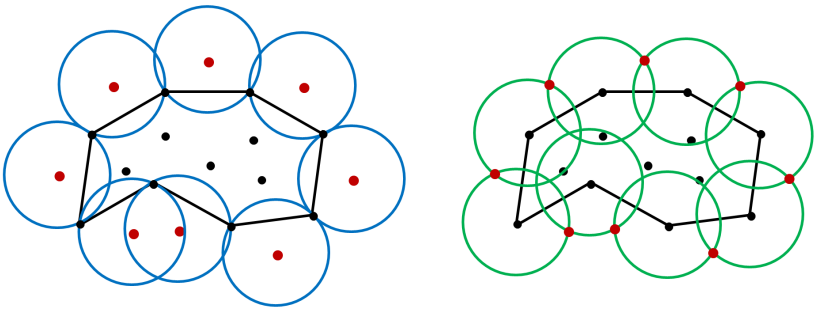
This section describes the beta mesh construction process. Figure 4.4 provides a 3D visualization of the algorithm running on a sphere. The algorithm proceeds in several steps, exemplified for 3D in Figure 4.4 and for 2D in Figure 4.8.

Step 1: beta vertices. Pure beta vertices are created: a single vertex for each alpha triangle, initially located at the center of the alpha circumscribing sphere. See Figure 4.5 for an illustration of the dual meaning of pure beta vertices. Together with the alpha graph connectivity the pure beta mesh is now defined, using pure beta vertices and pure beta faces. To be able to triangulate the beta faces late on, we create alphabeta vertices: a single vertex for each alpha graph face, initially



(a) (b) (c) (d)

Figure 4.4: Beta Mesh Algorithm Visualization (3D). (a) Starting with the "union of balls" surface, initial pure beta vertices (white dots) are positioned at the intersections of three or more balls. Pure beta faces, derived from the alpha graph, can be visualized as ball caps bounded by arcs of ball intersections. (b) Arcs are replaced by edges; note that face vertices are typically not coplanar. (c) Alphabeta vertices (red dots) initial position is set and face triangulation is performed. (d) The smoothing process repositions pure beta vertices at the weighted sum of (initial) pure beta vertices of adjacent faces, and alphabeta vertices at the weighted sum of the face (new) pure beta vertices. The smoothing result is presented, demonstrating improved mesh quality.



(a) (b)

Figure 4.5: Construction of Pure Beta Vertices. Pure beta vertices can be computed equivalently from the alpha shape disk centers, or from the union of balls intersections. Particles are shown in black. (a) Circumscribing circles (blue) associated with the directed alpha mesh edges (black) define the position of pure beta vertices (red) as the circle center. (b) Union of balls (green) define the position of pure beta vertices (red) as the intersection between two or more circles.

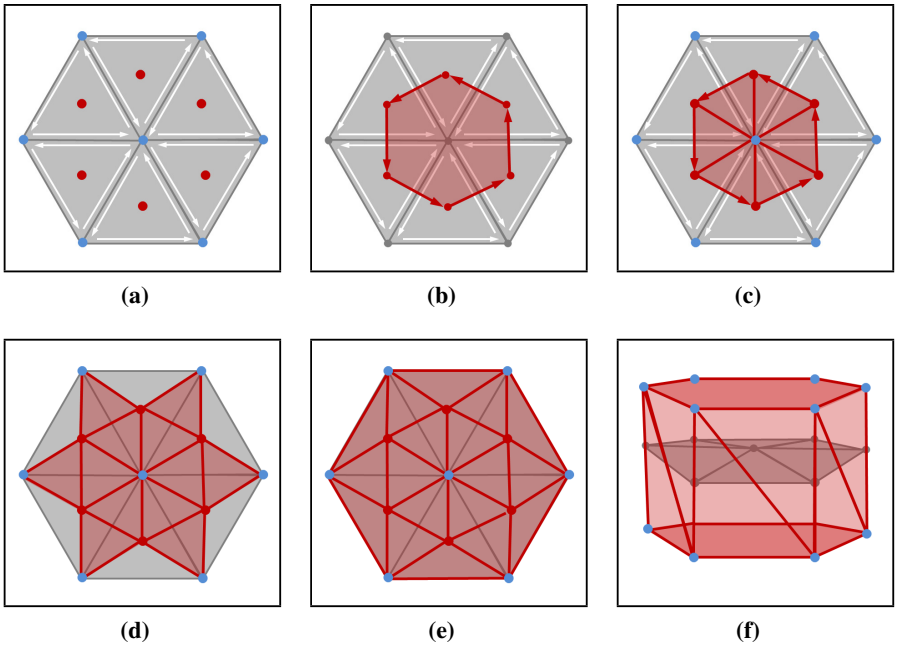


Figure 4.6: Beta Tessellation of a Thin Sheet. Presentation of alpha mesh/vertices (gray), beta mesh (red), pure beta vertices (red dots), alphabeta vertices (blue dots); top view is used except for (f) with side view. (a) Pure beta and alphabeta vertices. (b) Pure beta face (without alphabeta vertices). (c) Tessellation of the center face. (d) Tessellation of the boundary faces. (e) Complete the front/back tessellation. (f) Close the rim (side view).

positioned on the approximated plane passing through the face, projecting the alpha vertex to the plane. Computing the plane normal uses the weighted normals of adjacent alpha face edges. Last, external beta vertices are created from isolated alpha vertices and degenerated alpha mesh edges. They do not take part in the beta mesh, and are saved for the frame visualization step.

Step 2: tessellation. The tessellation process relies on the alpha graph: each closed alpha face defined a closed beta face with the same number of triangles. Each open alpha face with k triangles defines an open beta face with $k-1$ triangles. To close the rim of thin sheets we define new triangles that do not correlate with alpha mesh triangles. The tessellation process proceeds in several steps. First

we tessellate beta faces, starting with closed faces. Every alphabeta vertex connects with the pure beta vertices in its face, using the orientation defined in the alpha graph. Similarity boundary face triangulation is performed. All the triangles created so far are composed of one alphabeta vertex and two pure beta vertices. Then we complete the tessellation of the thin sheet rim. First we add a triangle connecting the two alphabeta vertices with the pure beta vertex on each side of the boundary edge. Finally we close the rim by tessellating quads of alphabeta vertices at the boundary edge with two triangles. Thin sheets in the alpha mesh result in flat sheets in the beta mesh of $2r$ thickness. See Figure 4.6 for an illustration of a thin sheet tessellation process.

Algorithm 11 Beta smoothing

Input: beta vertices $P = \{\mathbf{p}_i\}$, beta triangles $T = \{t_i\}$, beta faces $G = \{f_i\}$
Output: new beta positions $P' = \{\mathbf{p}'_i\}$

for all $\mathbf{p} \in P$ pure beta vertex **do**
  let $G^p = \{f_j\}$ be the set of faces of $\mathbf{p}$
  $\mathbf{p_{new}} \leftarrow \mathbf{0}$, $weight \leftarrow 0$
  for all $f_j \in G^p$ **do**
    for all $< \mathbf{p}_a, \mathbf{p}_b > \in f_j$ pure beta edge **do**
      $w \leftarrow dist(\mathbf{p}_a, \mathbf{p}_b)$
      $\mathbf{p}' \leftarrow \frac{\mathbf{p}_a + \mathbf{p}_b}{2}$
      $weight \leftarrow weight + w$
      $\mathbf{p_{new}} \leftarrow \mathbf{p_{new}} + w\mathbf{p}'$
  $\mathbf{p_{new}} \leftarrow \frac{\mathbf{p_{new}}}{weight}$
for all $\mathbf{p} \in P$ pure beta vertex **do**
  $\mathbf{p} \leftarrow \mathbf{p}_{new}$
for all $\mathbf{q} \in P$ alphabeta vertex **do**
  let f be the beta face of $\mathbf{q}$
  $\mathbf{q_{new}} \leftarrow \mathbf{0}$, $weight \leftarrow 0$
  for all $< q_a, q_b > \in f$ pure beta edge **do**
    $w \leftarrow dist(\mathbf{q}_a, \mathbf{q}_b)$
    $\mathbf{q}' \leftarrow \frac{\mathbf{q}_a + \mathbf{q}_b}{2}$
    $weight \leftarrow weight + w$
    $\mathbf{q_{new}} \leftarrow \mathbf{q_{new}} + w\mathbf{q}'$
  $\mathbf{q_{new}} \leftarrow \frac{\mathbf{q_{new}}}{weight}$
  $\mathbf{q} \leftarrow \mathbf{q_{new}}$

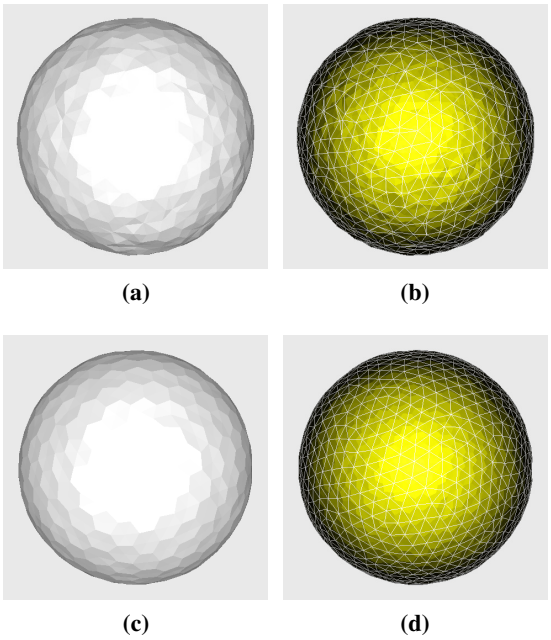


Figure 4.7: Beta Mesh Smoothing Step. Top: beta mesh when faces are connected to a weighted centroid. Bottom: the result after the smoothing step. Left: standard mesh representation. Right: front mesh triangles drawn in wire mode.

Step 3: smoothing. The smoothing process repositions the beta vertices but leaves the connectivity unchanged. First, the position of every pure beta vertex is recomputed as a weighted average of all the pure beta vertices in its faces. Then, the position of every alphabeta vertex is recomputed as a weighted average of the pure beta vertices in its face. A pseudocode is provided in Algorithm 11. Figure 4.7 demonstrates the improved mesh quality following the smoothing process.

Step 4: Resolving Beta/Alpha Mesh Intersections (optional). We propose detecting where the beta mesh intersects the alpha mesh and preventing those intersections, otherwise a particle could lie outside of the beta mesh. The algorithm goes through all the alpha faces, and checks if there is a triangle that intersects a beta triangle that belongs to the correlated beta face. If an intersection occurs it pushes the alphabeta vertex to the alpha vertex position. By construction a pure

beta vertex cannot cause such intersections, hence the alphabeta vertex position is the one that needs to be adjusted.

4.3.5 Temporal Coherence

The primary goal of the beta mesh surfacing algorithm is to maintain temporal coherence and yet achieve all the other desirable geometrical properties. Its construction relies both on the union of balls to benefit from its temporal coherence property, and on the alpha mesh to benefit from its geometrical characteristics. In this section we discuss the temporal coherence problems in the alpha mesh, and explain how the beta mesh construction overcomes these issues and maintains temporal coherence.

Alpha Mesh - Temporal Coherence

The union of balls is temporally coherent, and so is the alpha shape being its dual [26]. However when alpha shape arcs and sphere sub-sections are replaced with alpha mesh edges and triangles, temporal coherence is lost. We identify three scenarios of temporal coherence problems in the alpha mesh.

Edge flip. The same set of alpha vertices that exists in two consecutive frames may go through a topological change, reflected in an unsmooth mesh change. Consider for example four alpha vertices that form two non-planar triangles that share an edge. Particle movement on the next step can change the triangulation of these vertices and two new non-planar triangles are created on the opposite diagonal.

Vertex appearance/disappearance. Marginal particle movement can cause a new alpha vertex to appear or an existing vertex to disappear from the mesh, resulting in a major geometrical change. E.g. an isolated particle that is just outside of the circumscribing sphere joins the alpha mesh on the next step, see illustration in Figure 4.9 (a) (b). Despite the very small change in the particle position, a major geometric change of magnitude r in the worse case takes place in the alpha mesh.

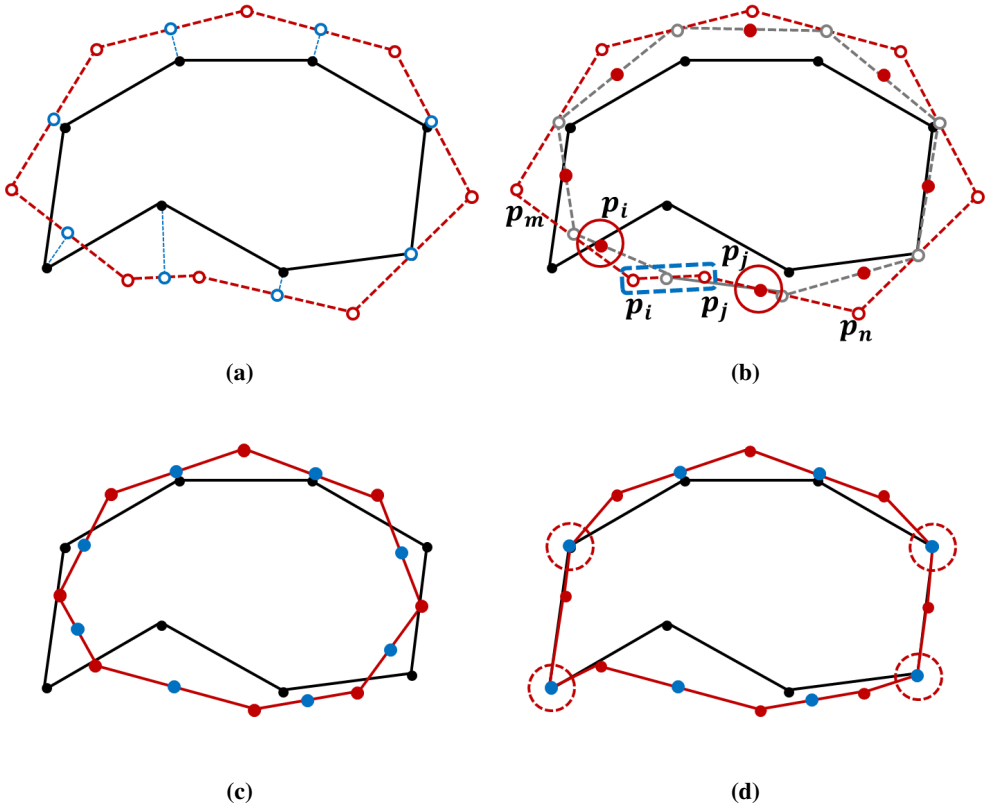


Figure 4.8: Beta Mesh Algorithm Illustration (2D). Alpha mesh is presented in black, beta mesh is presented in red. (a) Beta mesh with initial vertex positions, showing pure beta vertices (hollow red dots), alphabeta vertices (hollow blue dots), and beta mesh edges (dotted red lines). (b) Beta mesh smoothing first step: reposition pure beta vertices. Initial mesh is presented in hollow red dots and dotted red lines, alphabeta vertices are not presented. Face centroids (hollow gray dots) are used to compute the new pure beta vertices (solid red dots). Note: the smoothing process fixes short edges, see the initial edge $< p_i, p_j >$ highlighted in blue vs. the distance between the final p_i, p_j vertices highlighted in red circles. (c) Beta mesh after smoothing, showing edges (red lines), pure beta vertices (solid red dots), and alphabeta vertices (solid blue dots). (d) Beta mesh after resolving alpha/beta mesh intersections. Alphabeta vertices that were repositioned are highlighted.

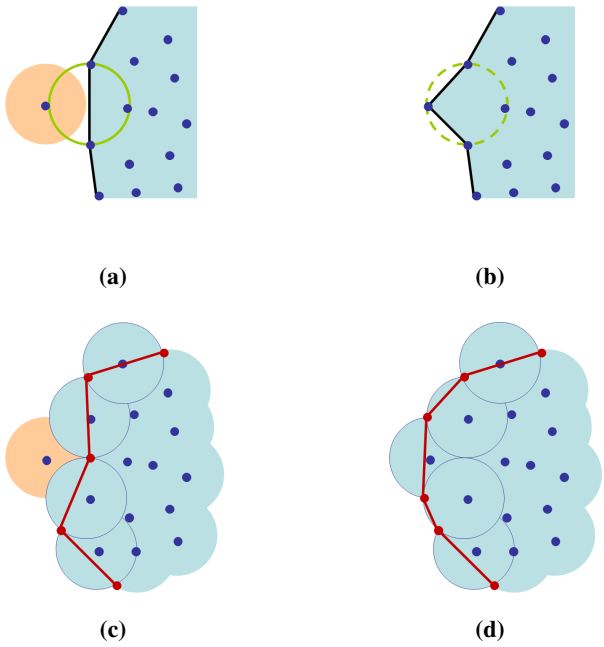


Figure 4.9: Water Drop Merge: Alpha Mesh vs. Beta Mesh. Upper row: Alpha mesh (black line). Lower row: Beta mesh (red line). Left: Step n, isolated particle highlighted (orange). Right: Step $n+1$, left particle becomes part of the mesh. (a) Isolated particle is outside of the alpha circumscribing circle (green). (b) The particle enters the alpha circumscribing circle (dotted green) and becomes part of the alpha mesh, causing a sharp mesh pop. (c) Isolated particle is outside of the beta mesh. (d) The particle becomes part of the beta mesh, inducing a smoother change.

Thin features. Alpha mesh thin features may disconnect/connect from one step to the next due to a marginal particle movement that eliminates/adds a vertex triplet from the thin triangles list. An opening in the thin sheet can then be formed in one step and close again in the next step.

While the first two scenarios create true problems in a surfacing technique, the third one potentially reflects an actual physical change (fluid separating or merging). A thin fluid layer that flows on a solid wall may open and close. A fluid drop

can merge and split from the fluid bulk, and nearly pushes/pops itself with very fast surface-tension-mediated dynamics which could be acceptably approximated with a temporal discontinuity. Our attention is therefore focused more on the first two scenarios, where temporal discontinuity is unacceptable.

Beta Mesh - Temporal Coherence

We examine next the beta mesh construction steps, and provide an informal explanation for why temporal coherence is maintained. Figure 4.10 demonstrates this discussion, and correlates to the numbered items below.

(a) The union of balls surface is temporally coherent.

(b) The intersections between three or more balls change smoothly in time and hence the positions of pure beta vertices are temporally coherent.

(c) Every two intersecting spheres on union of balls surface form a circular arc, or a complete circle in some cases. This network of arcs and circles changes smoothly in time.

(d) Beta faces can now be formed from a series of pure beta vertices that surround a sphere and are connected with circular arcs from the arcs network. The arcs geometry is temporally coherent as well as the induced graph/face connectivity. We can now replace the connectivity arcs between two pure beta vertices with edges, this construction still maintains temporal coherence.

(e) Alphabeta vertices are computed as a weighted sum of the face pure beta vertices, hence preserve temporal coherence.

(f) Hence triangulation of alphabeta vertices with pure beta vertices in its face also changes smoothly in time.

(g) The smoothing process recomputes pure beta vertex positions as a weighted sum of pure beta vertices in adjacent faces. The weighted sum is a temporally coherent mapping running on temporally coherent positions, therefor does not violate temporal coherence.

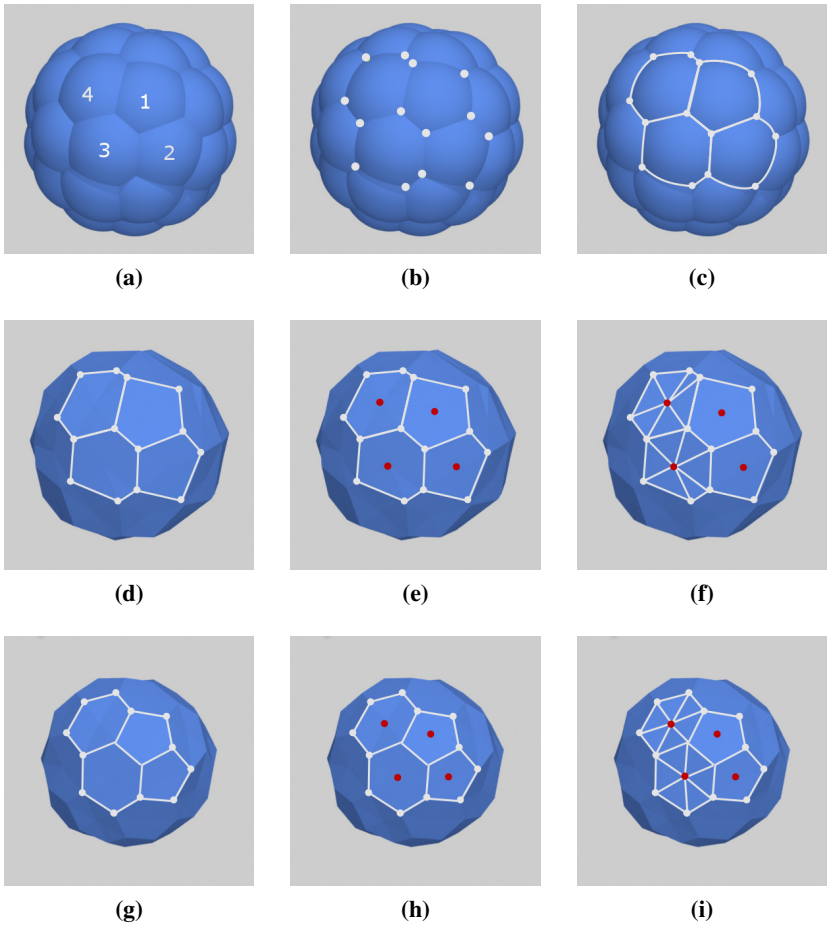


Figure 4.10: Temporal Coherence in the Beta Mesh Construction. Upper row: union of balls. Middle row: beta mesh before smoothing. Lower row: beta mesh after smoothing. (a) Union of balls, four front balls are highlighted. (b) Pure beta vertices are created: a pure beta vertex is an intersection of three or more balls. (c) A network of arcs is marked, each arc is an intersection of two balls. (d) Arcs convert to edges, forming pure beta faces. (e) Alphabeta vertices are created on the face's approximated plane. (f) Beta faces get tessellated. (g) Beta faces after smoothing: vertices repositioned, connectivity is left unchanged. (h) Alphabeta vertices in the smoothed mesh. (i) The smoothing process improves the mesh quality, notice the new triangulation of face-3 and face-4.

(h) Alphabeta vertices are recomputed as face centroids as weighted sum of the new pure beta vertices in each face; these are temporally coherent acts on temporally coherent positions, hence preserve temporal coherence.

(i) Last, triangulation of alphabeta vertices with the face pure beta vertices connects temporally coherent beta vertices and relies on a temporally coherent connectivity, hence it also changes smoothly in time.

To summarize, the beta mesh building blocks are temporally coherent, its vertices and the connectivity units are by construction temporally coherent, and therefore the resulting mesh changes smoothly in time.

Note that the circles/arcs network described above represents tendril features as a series of disconnected circles along the tendril. These circles lack intersection points, hence they do not impose pure beta vertices or beta faces. Consequently tendrils geometries (and similarly other alpha mesh degenerate features) are left outside of the beta mesh geometry in favor of keeping our temporally coherent construction rules.

Physical unsmooth changes in time are also reflected in the beta mesh, e.g. on thin sheet boundaries, or in merge/split of fluid drops or chunks. These changes however have better appearance in the beta mesh compared to the alpha mesh. Figure 4.9 illustrates a water drop merge, causing a sharp pop in the alpha mesh and a smoother change in the beta mesh.

4.4 Results and Discussion

Results of a 2D SPH dam break are shown in Figure 4.11. In our 2D implementation the graph analysis is consistent for simple volumetric regions and more complex regions with double-sided edges and thin features, and the stitching between different regions is done seamlessly. Beta mesh and alpha mesh intersections are resolved. The resulting beta mesh is smooth, and reflects alpha thin features.

Results of a 3D SPH dam break are presented in Figure 4.12. The top row (a)-(c) uses the union of balls particle rendering and provides the ground truth on how the surface should look like. The middle row (d)-(f) presents the alpha mesh surfacing, demonstrating an excellent representation of thin sheet in (f). Some craters

are seen in the surface that may indicate high sensitivity to particle uneven distribution, or to the alpha radius chosen (here $r = 1.5dx$). In the lower row (g)-(i) the beta mesh surfacing is presented. The surfaces are much smoother than the alpha mesh surfaces. Thin sheets are mostly captured, though the boundaries implementation may not yet be fully satisfying. The craters visible in (i) reflect our major open problem, in the stitching of double-sided triangles to the fluid volumetric surface. The desired temporal coherence is achieved, demonstrated in the animations. The frame data consists of 110k particles and the surfacing run time is 19 seconds per frame that is mostly taken for the alpha mesh construction.

Figure 4.13 shows the surfacing of a rotating cube FLIP simulation. The frame data consists of 170k particles. Particles are unevenly distributed, which is reflected in unsmooth surfaces and long run times (up to 6 hours per frame), that again is mostly taken for alpha mesh processing. Near the surface, in this example, the average number of particles per grid cell was much higher, leading to a significantly higher constant in the $O(n)$ run time. Figures (a), (d) and (g) are most interesting for examination, demonstrating exceptionally good tendril features representation in the alpha mesh. The beta mesh is currently not able to reproduce the desired tendril look as is indicated in our open problems list. The particle inhomogeneous distribution is visible in (b), (e), (h), reflected in unsmooth surfaces in all three representations, nevertheless the beta produces much smoother surfaces compared to the alpha mesh.

4.4.1 Open Issues

The critical area to improve in our beta mesh construction is the graph interpretation of complex geometrical scenarios in the alpha mesh that may appear when double-sided alpha triangles connect to the rest of the mesh. Examples are air bubbles trapped in the fluid resulting in holes in the surface, stitching between thin sheets and volumetric sections, or thin fluid features connecting to each other in nontrivial ways.

Another aspect that we would like to further improve is thin sheet boundary handling. Extending the surface beyond the boundary sphere center could assist towards better temporal coherence there.

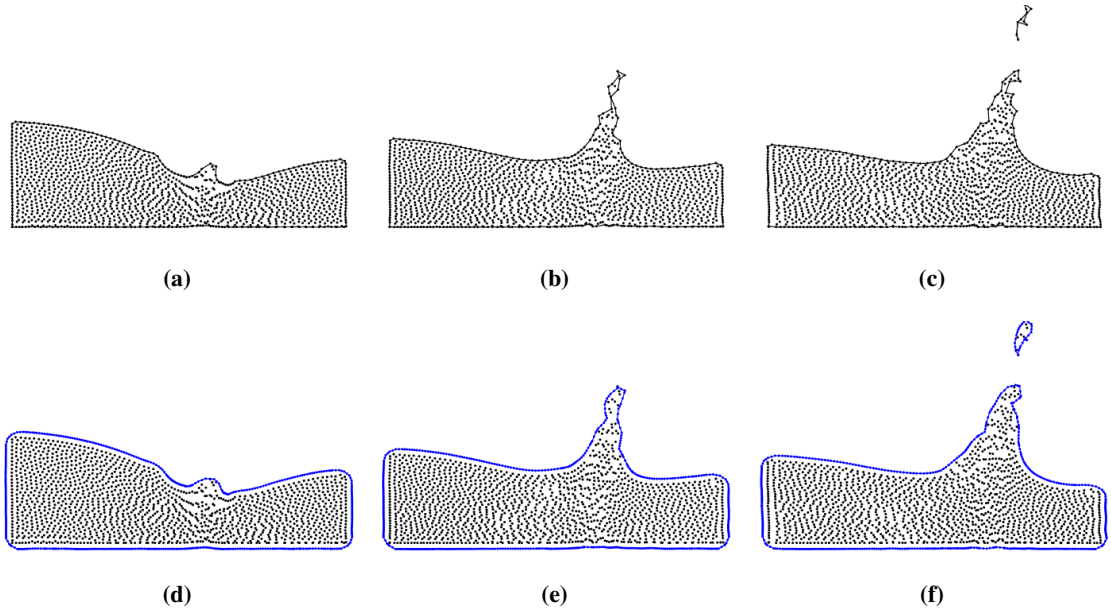


Figure 4.11: 2D Dam Break. Upper row (a)-(c): Alpha mesh. Lower row (d)-(f): Beta mesh. (equivalent frames are presented)

Fluid tendrils are reflected in the 3D alpha mesh as degenerate edges, are presently not well captured in the beta mesh. Degenerate alpha features lend themselves to become external beta vertices, and thread-like alpha geometry may shrink its volume in the beta mesh. Better representation will increase the usability of the beta mesh technique.

Our generic smoothing procedure shows excellent results, however in some cases it may fail its designed task. A simple example is a fluid tetrahedron that will shrink after smoothing into a point. Further analysis may be required to cover such cases.

The alpha mesh procedure could also be improved. Currently our sphere criterion uses power and square root computation, which are sensitive to rounding issues. Alternative checks could be examined to avoid this sensitivity and guarantee robustness.

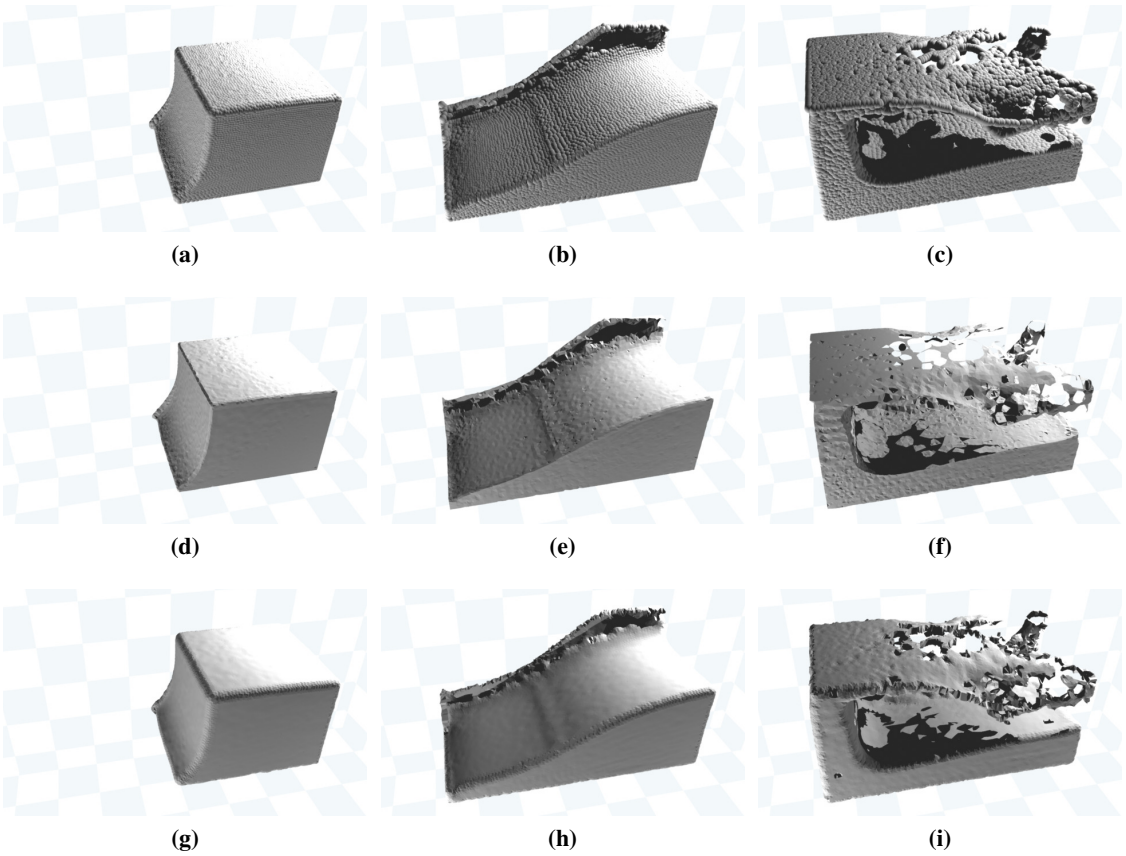


Figure 4.12: 3D Dam Break. Upper row (a)-(c): Union of Balls. Middle row (d)-(f): Alpha mesh. Lower row (g)-(i): Beta mesh. (equivalent frames are presented)

4.5 Conclusions

In conclusion, two surfacing techniques are presented in this work:

The alpha mesh is a well known technique, which we first use and analyze for animation purposes and provide an efficient new algorithm to compute it.

The beta mesh is a novel surfacing construction that we introduce, which blends ideas from the union of balls surfacing and the alpha mesh construction.

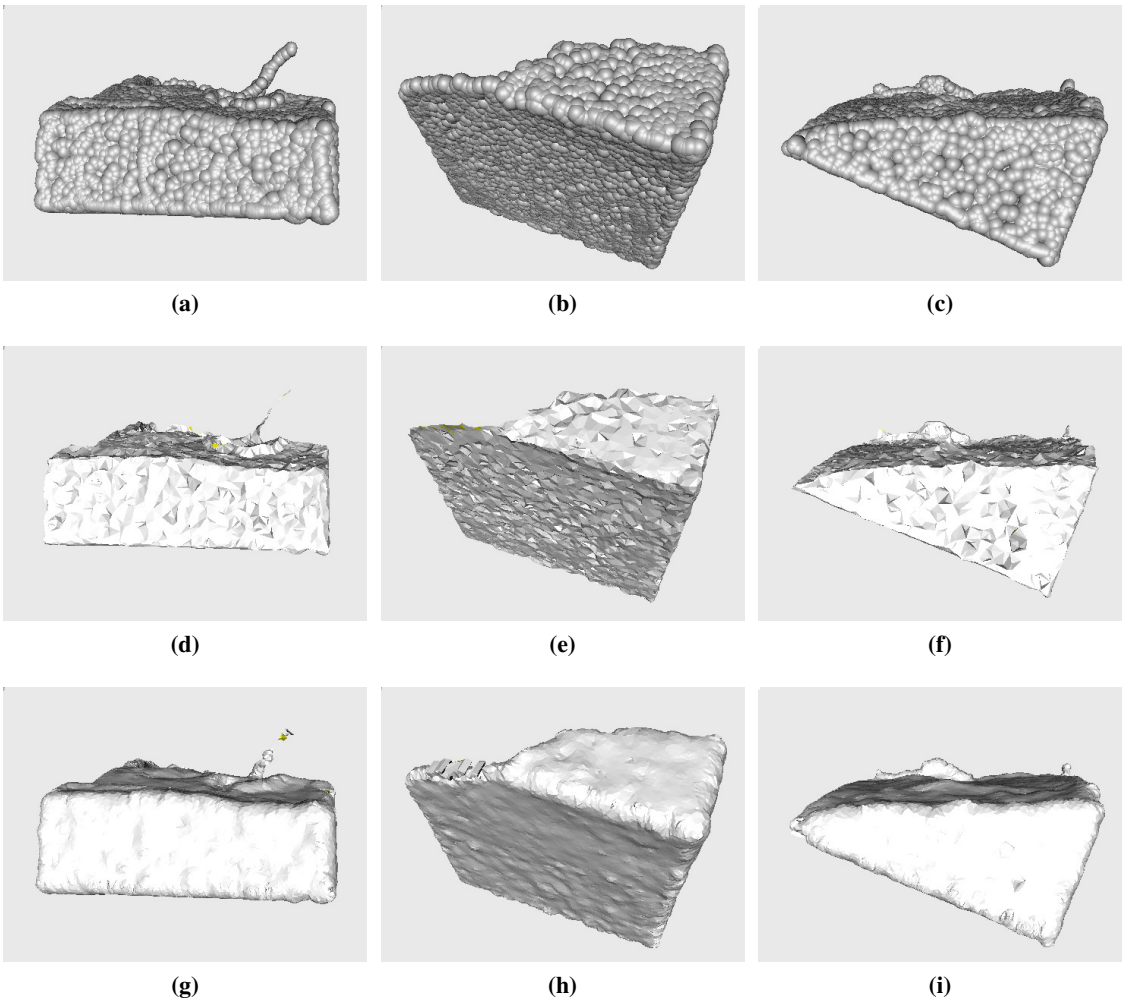


Figure 4.13: Rotating Cube. Upper row (a)-(c): Union of Balls. Middle row (d)-(f): Alpha mesh. Lower row (g)-(i): Beta mesh. (equivalent frames are presented)

The alpha mesh representation of thin sheets and the tendril features are of significance for animation purposes, and no other surfacing method known to us can similarly reproduce these features. The temporal coherence issues of the alpha mesh are being considered in this work. Bringing the alpha mesh to the animation regime, and illustrating its pros and cons in this context, is of significant relevance to the field.

The beta mesh addresses an important surfacing requirement – the surface temporal coherence, which had little attention in related research works. In addition to producing smooth transitions of the surface in time, the method targets conforming to important geometrical requirements such as reflecting flat surfaces in the same algorithmic framework.

The beta mesh addresses its primary goal and its demonstrated benefits look very promising. Temporal coherence is achieved, smooth surfaces are well demonstrated, and fine features are preserved. Nevertheless, further improvements should take place to make it a useful and robust tool for animation purposes, especially on 3D complex geometrical graph sections. Handling problems and features outlined in the open issues section could make it usable for handing a broad range of phenomena.

In summary, we believe that the ideas presented in this work could serve as a basis for establishing useful particle skinning techniques in the future, that could encompass the non-trivial combination of the three desired surface requirements: temporal coherence, spatial smoothness, and fine features.

Chapter 5

Conclusion

5.1 Sub-Grid Turbulence

Chapter 2 introduces a simple and efficient model for dynamically adding sub-grid turbulence details using a procedural post-processing of a large-scale simulation motion. A similar sub-grid approach for smoke simulation was introduced concurrently to our work by Kim et al. [42], and later that year also by Narain et al. [51]. An analogous turbulence synthesis method was later implemented on a GPU for fire simulation by Horvath and Geiger [38]. Our smoke model significantly improves the visual realism because it not only adds sub-scale details, but also transports and diffuses turbulent energy in multiple scales both spatially and temporally.

Our approach can obviously influence the computer graphics industry. Visual effects tools can now embed this efficient way of adding small scale detail to existing large scale smoke simulators. Interactive applications can implement a GPU version similar to Horvath and Geiger [38] and include realistic turbulence animations in games.

Future directions to this work can be classified in two groups. The first is to apply sub-grid turbulence details to animate other phenomena, e.g. a water fall animation. The second class is to further develop our turbulence model, e.g. to examine how to capture onset turbulence for computer animation purposes.

5.2 Boundary Handling in SPH

Chapter 3 introduces a new ghost fluid approach for free surface and solid boundary conditions in SPH liquid simulations. The first important impact of our approach is addressing a well known problem in SPH: particle deficiency at the interface with air and solids. With Ghost-SPH surface-tension like artifacts are avoided and instead the fluid can perform free motion in air and natural cohesion with solids. Our technique supports complex solid boundaries, either static or dynamic, and can be used for a wide variety of scenes. We also provide a new simpler form of artificial viscosity based on XSPH that uses kernel smoothing of neighborhood velocities and is more intuitive and easier to tune.

Our Ghost-SPH boundary conditions treatment could be extended in the future to other pure particle methods, that as well are currently restricted by particle voids in air and solid interface.

We also introduce a new sampling algorithm that supports correct boundary conditions in air and solids. It produces efficient surface-tight sampling, that can be used statically for the initial simulation particle seeding in fluid and solid arbitrary geometries, or dynamically during simulation to emit new fluid particles and air particles around the liquid. Our algorithm is generic and can be used with other Lagrangian methods as well as for different computer graphics applications.

Particles presence on the other side of the interface opens up new directions to extend the Ghost SPH method in future work. We plan to examine a more accurate treatment of surface tension, e.g. for modeling of droplet closeup using SPH. Two-way coupling with solids or other fluids is another direction to explore.

5.3 Temporally Coherent Surface Reconstruction from Particles

Chapter 4 suggests a novel approach for a temporally coherent surface reconstruction technique. We use the union of balls surface as building blocks for the surface reconstruction procedure to leverage temporal coherence. We rely on the good geometrical properties of the alpha mesh to capture fine details in the flow. We run a post-processing smoothing step to reflect smooth surfaces. Combining a temporally coherent surfacing technique with desired geometrical properties is a very

promising direction that tackles an unaddressed important topic in Lagrangian fluid simulation.

In the future we would like to primarily address complex geometrical scenarios in the beta mesh processing to be able to robustly handle any simulation data. In addition, we wish to further explore other open issues that were outlined.

Bibliography

[1] B. Adams, P. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007. → pages 31, 57

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 21–28, 2001. → pages 57

[3] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266, 2001. → pages 58

[4] A. Angelidis and F. Neyret. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, 2005. → pages 13

[5] C. Batty. *Simulating Viscous Incompressible Fluids with Embedded Boundary Finite Difference Methods*. PhD thesis, The University Of British Columbia, 2010. → pages 56

[6] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007. → pages 38

[7] M. Becker and M. Teschner. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 63–72, 2007. → pages 31, 34

[8] M. Becker, M. Ihmsen, and M. Teschner. Corotated SPH for deformable solids. In *Proceedings Eurographics Workshop on Natural Phenomena*, pages 27–34, 2009. → pages 53

[9] M. Becker, H. Tessendorf, and M. Teschner. Direct forcing for Lagrangian rigid-fluid coupling. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):493–503, 2009. → pages 31

[10] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999. → pages 58

[11] H. Bhatacharya, Y. Gao, and A. Bargteil. A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 17–24, 2011. → pages 57

[12] J. F. Blinn. A generalization of algebraic surface drawing. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1982)*, 16(3):273–, 1982. → pages 2, 5, 56

[13] J. Bonet and S. Kulasegaram. A simplified approach to enhance the performance of smooth particle hydrodynamics methods. *Applied Mathematics and Computation*, 126(2-3):133–155, 2002. → pages 31

[14] L. Boyd and R. Bridson. MultiFLIP for energetic two-phase fluid simulation. *ACM Transactions on Graphics*, 31(2):16:1–16:12, Apr. 2012. → pages 53, 56

[15] J. U. Brackbill and H. M. Ruppel. FLIP: a method for adaptively zoned, particle-in-cell calculuations of fluid flows in two dimensions. *Journal of Computational Physics*, 65:314–343, 1986. → pages 3

[16] R. Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, 2007. → pages 32, 38, 41

[17] R. Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008. → pages 2

[18] R. Bridson and M. Müller-Fischer. Fluid simulation, 2007. ACM SIGGRAPH 2007 Courses. → pages 9

[19] R. Bridson, J. Hourihan, and M. Nordenstam. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007. → pages 12, 19, 20

[20] N. Chentanez and M. Müller. A multigrid fluid pressure solver handling separating solid boundary conditions. In *Proceedings of the 2011 ACM*

SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11, pages 83–90, 2011. → pages 38

[21] A. Colagrossi and M. Landrini. Numerical simulation of interfacial flows by Smoothed Particle Hydrodynamics. *Journal of Computational Physics*, 191 (2):448–475, 2003. → pages 31

[22] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1), 1986. → pages 32

[23] R. L. Cook and T. DeRose. Wavelet noise. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):803–811, 2005. → pages 21

[24] M. Desbrun and M.-P. Cani. Smoothed Particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76, 1996. → pages 3, 53

[25] D. Dunbar and G. Humphreys. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3), 2006. → pages 32

[26] H. Edelsbrunner. The union of balls and its dual shape. In *Proceedings of the ninth annual symposium on Computational geometry*, SCG '93, pages 218–231, 1993. → pages 54, 56, 73

[27] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994. → pages 54, 56, 61, 62

[28] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4): 551–559, 1983. → pages 54, 56, 60, 61

[29] R. Fedkiw. Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the Ghost Fluid Method. *Journal of Computational Physics*, 175:200–224, 2002. → pages 31

[30] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the Ghost Fluid Method). *Journal of Computational Physics*, 152:457–492, 1999. → pages 4, 31

[31] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 15–22, 2001. → pages 12, 13

[32] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 23–30, 2001. → pages 56

[33] M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *In Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation*, pages 3–15, 1995. → pages 13

[34] R. A. Gingold and J. J. Monaghan. Smoothed Particle Hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, 1977. → pages 3, 30

[35] F. H. Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. In *Experimental arithmetic, high-speed computations and mathematics*, 1963. → pages 2

[36] F. H. Harlow and P. I. Nakayama. Turbulence transport equations. *Physics of Fluids*, 10(11):2323–2332, 1967. → pages 11

[37] J.-M. Hong, H.-Y. Lee, J.-C. Yoon, and C.-H. Kim. Bubbles alive. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3): 48:1–48:4, 2008. → pages 53

[38] C. Horvath and W. Geiger. Directable, high-resolution simulation of fire on the GPU. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3):41:1–41:8, 2009. → pages 84

[39] M. Ihmsen, N. Akinci, M. Gissler, and M. Teschner. Boundary handling and adaptive time-stepping for PCISPH. In *Proceedings of the Seventh Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2010*, pages 79–88, 2010. → pages 31

[40] R. Keiser, B. Adams, L. J. Guibas, P. Dutré, and M. Pauly. Multiresolution particle-based fluids. Technical Report 520, ETH Zurich, 2006. → pages 31

[41] B. Kim, Y. Liu, I. Llama, and J. Rossignac. FlowFixer: using BFECC for fluid simulation. In *Proc. Eurographics Workshop on Natural Phenomena*, 2005. → pages 12

[42] T. Kim, N. Thürey, D. James, and M. Gross. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):50:1–50:6, 2008. → pages 12, 84

[43] J. Kniss and D. Hart. Volume effects: modeling smoke, fire, and clouds, 2004. Section from ACM SIGGRAPH 2004 Courses, *Real-Time Volume Graphics*, http://www.cs.unm.edu/ jmk/sig04_modeling.ppt. → pages 11, 19

[44] T. Lenaerts, B. Adams, and P. Dutré. Porous flow in particle-based fluid simulations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):49:1–49:8, 2008. → pages 53

[45] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977. → pages 3, 30, 31

[46] J. J. Monaghan. On the problem of penetration in particle methods. *Journal of Computational Physics*, 82:1–15, 1989. → pages 32, 34

[47] J. J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110:399–406, 1994. → pages 5, 30, 31, 33, 34

[48] J. J. Monaghan. Smoothed Particle Hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 2005. → pages 3, 29, 32

[49] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, 2003. → pages 3, 30, 31, 53, 57

[50] M. Müller, B. Solenthaler, and R. Keiser. Particle-based fluid-fluid interaction. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 237–244, 2005. → pages 31, 53

[51] R. Narain, J. Sewall, M. Carlson, and M. C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2008)*, 27(5): 166:1–166:8, 2008. → pages 84

[52] F. Neyret. Advected textures. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 147–153, 2003. → pages 12, 19

[53] S. I. Park and M. J. Kim. Vortex fluid for gaseous phenomena. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, 2005. → pages 13

[54] K. Perlin. An image synthesizer. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1985)*, 19(3):287–296, July 1985. → pages 20

[55] K. Perlin and F. Neyret. Flow noise. In *ACM SIGGRAPH 2001 Sketches and Applications*, page 187, 2001. → pages 12, 14, 19

[56] S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2005. → pages 9

[57] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum (Proceedings of Eurographics 2003)*, 22(3):401–410, 2003. → pages 56

[58] N. Rasmussen, D. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22:703–707, 2003. → pages 11

[59] W. T. Reeves. Particle systems - technique for modeling a class of fuzzy objects. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1983)*, 17(3):359–375, 1983. → pages 2, 55

[60] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 1–7, 2008. → pages 6, 53

[61] H. Schechter and R. Bridson. Ghost SPH for animating water. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, 31(4): 61:1–61:8, 2012. → pages 7

[62] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, pages 910–914, 2005. → pages 13

[63] M. Shinya and A. Fournier. Stochastic motionmotion under the influence of wind. *Computer Graphics Forum (Proceedings of Eurographics 1992)*, 11 (3):119–128, 1992. → pages 3, 11

[64] K. Sims. Particle animation and rendering using data parallel computation. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):405–413, 1990. → pages 55

[65] F. Sin, A. W. Bargteil, and J. K. Hodgins. A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 247–255, 2009. → pages 56

[66] B. Solenthaler and M. Gross. Two-scale particle simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4): 81:1–81:8, 2011. → pages 31

[67] B. Solenthaler and R. Pajarola. Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–218, 2008. → pages 31, 53

[68] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009)*, 28(3), 2009. → pages 32

[69] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, 1999. → pages 12

[70] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 369–376, 1993. → pages 3, 11

[71] R. Tam and W. Heidrich. Computing polygonal surfaces from unions of balls. In *Proceedings of the Computer Graphics International*, CGI '04, pages 86–92, 2004. → pages 58

[72] H. Tennekes and J. L. Lumley. *A first course in turbulence*. MIT Press, 1972. → pages 9

[73] G. Turk. Re-tiling polygonal surfaces. *ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH 1992)*, 26(2):55–64, 1992. → pages 32

[74] B. W. Williams. Fluid surface reconstruction from particles. Master's thesis, The University Of British Columbia, February 2008. → pages 57

[75] L. Yaeger, C. Upson, and R. Myers. Combining physical and visual simulation—creation of the planet jupiter for the film 2010. In *Proc. ACM SIGGRAPH*, pages 85–93, 1986. → pages 13

[76] J. Yu and G. Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 217–225, 2010. → pages 57

[77] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, pages 965–972, 2005. → pages 3, 12, 17, 53, 57