Efficient and Detailed Simulation of Fluids for 3D Computer Graphics

by

François DAGENAIS

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, SEPTEMBER 6TH 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Prof. Eric Paquette, Thesis supervisor
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

Prof. François Morency, Chair, Board of Examiners
Département de génie mécanique, École de technologie supérieure

Prof. Luc Duong, Member of the Jury
Département de génie logiciet et des technologies de l'information, École de technologie supérieure

Prof. Christopher Batty, External Independent Examiner
David R. Cheriton School of Computer Science, University of Waterloo

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON JULY 31TH 2025

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

The work described in chapter 2 and in chapter 3 was funded by Mokko Studio, Digital District Canada, MITACS, Prompt, and ÉTS. We thank the former R&D team and employees at Mokko Studio and Digital District for their feedback and involvement throughout these projects. The work described in chapter 4 was funded by OSSim Technologies, NSERC, and ÉTS. We thank employees from OSSim Technologies for their valuable involvement in this project. We would also like to thank David Mould from Carleton University for his involvement in these projects. We thank Julián Guzmán, Valentin Vervondel, Jonathan Gagnon and Bruno Roy from ÉTS for their involvement throughout these projects. Finally, we thank SideFX for providing Houdini licenses for these research projects.

Simulation détaillée et efficace
de phénomènes liquides et granulaires en infographie 3D

François DAGENAIS

RÉSUMÉ

Les simulations physiques sont souvent utilisées en infographie afin de reproduire le mouvement de matériaux complexes tels que les liquides et la neige. La résolution et la qualité visuelle de ces simulations sont généralement limitées par la mémoire vive et les temps de calcul disponibles. Dans cette thèse, nous introduisons trois approches novatrices dont l'objectif est la production de simulations visuellement détaillées qui nécessitent moins de mémoire et de temps de calcul. La première approche utilise un maillage surfacique explicite qui s'ajuste selon la surface d'une simulation de liquide par particules. Elle utilise une projection qui préserve les détails de la surface afin de prévenir un décalage entre celle-ci et les particules. De plus, nous introduisons une nouvelle opération de correction de topologie qui garantit que la topologie de la surface explicite demeure cohérente avec les particules. Cette approche rend possible la conservation de détails visuels fins sur la surface avec un nombre moindre de particules, ce qui améliore considérablement les temps de calcul. Elle a été testée avec succès sur des simulations de type *smoothed-particle hydrodynamics (SPH)* et *fluid implicit particle (FLIP)*. De plus, elle peut être exécutée après que la simulation ait entièrement été exécutée, ce qui la rend idéale pour le processus de création itératif typique des studios d'effets visuels. La seconde approche vise la simulation d'une couche de neige au sol qui interagit avec des personnages et objets animés. Elle décompose le volume de neige en trois composants: la couche de base (neige intacte), la neige dynamique, et les particules plus fines de neige. Cette décomposition permet d'utiliser une représentation volumétrique pour la couche de base, ainsi qu'une simulation de fluide semi-Lagrangienne pour les particules fines. Celles-ci sont beaucoup moins gourmandes en mémoire et temps de calculs que la simulation par particules utilisée pour la neige dynamique. Lorsque les personnages interagissent avec la neige intacte, cette dernière est transférée de la couche de base vers la simulation de neige dynamique. À leur tour, les particules dynamiques projetées dans les airs ajoutent de la neige dans la simulation de neige fine. Cette décomposition permet de simuler de plus grands volumes de neige tout en concentrant la mémoire et les temps de calcul où ils sont le plus utiles. Finalement, la troisième approche simule et visualise efficacement de petits volumes de liquide en temps réel à l'aide d'une simulation 2.5D de colonnes de liquide. Elle introduit un algorithme novateur de construction de surface qui gère beaucoup mieux les surplombs et les changements topologiques, ainsi qu'un nouveau modèle de viscosité basé sur la physique qui est à la fois stable et rapide. Pour y arriver, ce modèle repose sur des hypothèses qui permettent de simplifier les équations physiques sous-jacentes. Bien que ces hypothèses diminuent la précision de ce modèle, nos résultats démontrent qu'il peut reproduire des comportements très proches de ceux obtenus à l'aide d'une simulation hors-ligne 3D plutôt que 2.5D, et utilisant le modèle complet plutôt que le modèle simplifié, ce qui nécessite beaucoup plus de temps de calculs.

Mots-clés: Infographie, 3D, Simulation physique, Fluides, Neige, Suivi de surface, Temps réel, Liquides peu profonds, Viscosité

Efficient and Detailed Simulation of Fluids for 3D Computer Graphics

François DAGENAIS

ABSTRACT

Physical simulations are often used in the field of computer graphics to replicate the motion of complex materials such as liquids and snow. The resolution and visual quality of these simulations are generally limited by the available memory and computation times. In this thesis, we introduce three novel approaches that aim to provide visually detailed simulations with reduced memory requirements and computation times. The first approach uses an explicit mesh surface that follows a particle-based simulation of a liquid. It uses a detail-preserving projection to prevent drifting of the surface from the particles. Additionally, we introduced a new topology matching stage which ensures that the surface topology remains consistent with the particles. This approach makes it possible to have a surface with finer visual details on top of a simulation with fewer particles, thus faster to compute. This approach has been tested with both *smoothed-particle hydrodynamics (SPH)* and *fluid implicit particle (FLIP)* simulations with success. Furthermore, it can be run after the whole simulation has been computed, making it ideal for the typical iterative process of visual effects studios. The second approach targets the simulation of a layer of snow on the ground that interacts with animated characters and objects. It decomposes the snow volume into three components: the base layer (untouched snow), dynamic snow, and finer particles of snow. This decomposition allows using a lightweight volumetric representation for the base layer, and a more efficient Semi-Lagrangian fluid simulation for the finer particles. As the characters interact with the snow, it is transferred from the base layer to the dynamic snow, which relies on a memory and computation expensive particle-based simulation. In turn, as dynamic snow particles are shoved in the air, density is added in the finer snow simulation. This decomposition allows us to simulate much larger volumes of snow with more details by focusing memory and computation times where it is needed the most. The third approach efficiently simulates and visualizes small-scale liquids in real-time using a 2.5D simulation of columns of liquid. It introduces a novel surface construction algorithm that better handles overhangs and changes in topology, as well as a new physically-based model for viscosity that is both stable and fast. This model relies on assumptions that enable simplifications of the underlying physical equations. While these assumptions lower the model's accuracy, our results showed that we can replicate behaviors close to that of much more computationally expensive 3D offline simulations that is based on the full viscosity model.

Keywords: Computer Graphics, 3D, Physical simulation, Fluids, Snow, Surface tracking, Real-time, Shallow liquid, Viscosity

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

APIC	Affine Particle-In-Cell
CFL	Courant–Friedrichs–Lewy
CPU	Central Processing Unit
CSG	Constructive Solid Geometry
CUDA	Compute Unified Device Architecture
DEM	Discrete Element Method
FEM	Finite Element Method
FLIP	FLuid Implicit Particle
GPU	Graphics Processing Unit
MICCG(0)	Modified Incomplete Cholesky Conjugate Gradient, Level Zero
MPM	Material Point Method
PIC	Paricle-In-Cell
RAM	Random-Access Memory
SDF	Signed Distance Field
SOR	Successive Over-Relaxation
SPH	Smoothed Particle Hydrodynamics
SSE	Shallow Sand Equations
SWE	Shallow Water Equations
VFX	Visual effects

VP	Virtual Pipes
XPBI	EXtended Position-Based Inelasticity

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

α_{ij}	Matrix entry at row i and column j (chapter 2)
β	Threshold value of the SOR solver stopping criterion (chapter 2)
β	Air resistance coefficient (chapter 3)
Γ	Implicit representation of the explicit mesh surface (chapter 2)
δ_i	Initial distance to the surface of vertex i (chapter 2)
ζ	Friction parameter (chapter 4)
ν	Kinematic viscosity (chapter 4)
ρ	Density of the liquid (chapter 4)
τ	Threshold for topological changes detection (chapter 2)
ϕ	Discretized implicit function (chapter 2)
ψ	Difference field (chapter 2)
ω	Relaxation factor of the SOR solver (chapter 2)
ω_{opt}	Optimal value of ω (chapter 2)
ω	Fraction of flux retained per unit time (chapter 4)
A	Cross-section area of a virtual pipe (chapter 4)
b_i	Base (terrain) height of cell i (chapter 4)
C	Cross-sectional area of the liquid's flow (chapter 4)
$d_{overlap,i}$	Predicted overlap between liquid of column i and the terrain on top (chapter 4)
$d_{total,i}$	Total liquid height change in cell i from incoming fluxes (chapter 4)

$f_{i,j}$	Flux between cells i and j (chapter 4)
g	Gravitational acceleration (chapter 3 and chapter 4)
$\mathbf{g}$	Gravity vector (chapter 4)
h	Cell size of grid Γ (chapter 2)
h_i	Liquid height of cell i (chapter 4)
$h_k^{\min}$	Minimum liquid surface height (chapter 4)
H	Liquid's depth (chapter 4)
K_i^{min}	Scaling factor for outflow fluxes in column i (chapter 4)
K_i^{max}	Scaling factor for inflow fluxes in column i (chapter 4)
l	Length of a virtual pipe (chapter 4)
$\min_i$	Lower bound of cell i (chapter 4)
$\max_i$	Upper bound of cell i (chapter 4)
N	Grid dimension (chapter 2)
N	Number of animation samples (chapter 3)
N	Simulation grid dimensions (chapter 4)
$\mathcal{N}_i$	Number of neighbors of cell i (chapter 2)
$\mathbf{n}$	Normal of a mesh vertex (chapter 2)
$\nabla\mathbf{p}_j$	Position update of particle i (chapter 2)
p	Pressure (chapter 4)
r	Influence radius (chapter 2)

r	Particles radius (chapter 3)
t	Time (chapter 3)
Δt	Time step (all chapters)
$\mathbf{u}$	Velocity (chapter 4)
$u_{i,j}$	Speed of the flow between cells i and j (chapter 4)
$u^p_{i,j}$	Velocity profile of the flow between cells i and j (chapter 4)
$\mathbf{v}_i$	Velocity of particle i (chapter 3)
$\mathbf{v}^*_i$	Modified velocity of particle i (chapter 3)
W_{ij}	Weighting function of particles i and j (chapter 2)
$\mathbf{x}_i$	Position of particle i (chapter 2)
$\mathbf{x}^{proj}_i$	Projection of $\mathbf{x}_i$ unto $\phi = \delta_i$ (chapter 2)
$\mathbf{x}^*_i$	Projection of $\mathbf{x}_i$ unto $\phi = 0$ (chapter 2)
Δx_ϕ	Cell size of grid ϕ (chapter 2)
Δx	Simulation grid cells size (chapter 4)

INTRODUCTION

An extensive amount of work in the field of computer graphics aims to visually replicate the world around us in order to bring to life virtual environments and characters on our screens. This includes, for example, imitating how light travels and interacts with different surfaces, how people and animals move, and even how complex materials such as water, smoke, and snow behave and look. The latter offers a particular challenge to animate, as the behavior of such materials is quite complex and very difficult to animate by hand. For that reason, physically-based simulations are generally used. However, these simulations often require a very high amount of data and computation times to accurately replicate the complex behavior of such materials. Capturing finer visual features as well as small-scale motions requires smaller simulation elements, which means a much larger amount of data is required to simulate a given volume of material. Furthermore, smaller simulation elements generally imply smaller time increments in the simulation in order to preserve stability and accuracy of the simulation. These two factors combined contribute to making highly detailed and realistic simulations of such materials very computationally expensive and also memory-intensive. As a result, the visual quality and realism of such animations are often limited by the memory resources and processing time available to run the simulation.

In the industry of visual effects (VFX) and feature animations, i.e., post-production for movies, television series, advertisements, etc., the final images are generated long before the viewer sees them. As a result, the time budget for the process of generating these images, also known as *rendering*, and for running fluids simulations as well is very large. Artists working on such projects generally have access to a render farm, i.e., a computer cluster dedicated to process computer graphics related tasks, which can significantly speed up those tasks by distributing the calculation on multiple dedicated machines. However, despite having access to a large amount of computation resources and a large time budget, computation times remain very important. Because these simulations must generally adhere to a specific artistic vision, multiple iterations

may be required to achieve the desired result. Smaller computation times hence mean being able to run more iterations to improve the result. As a rule of thumb, an artist who starts a simulation before the end of their shift expects it to be fully completed by the time they come back the next morning. As the expectations in terms of visual quality in visual productions keep increasing, the tools the artists use must always push the limits to create virtual fluids with a high amount of visual details. Luckily, in this industry, visual plausibility is favored over physical accuracy, which allows the use of simpler equations and approximations that are faster to compute. Other industries, such as the gaming industry, virtual reality, and medical training simulators, have a substantially smaller time budget. Everything must happen in real-time as the user provides inputs into the system. In such real-time systems, we often aim to be able to generate at least 60 images per second ($\sim$ 16ms per image), which includes all calculations such as handling user inputs, game/system logic, running the simulations, rendering the images, playing sounds, and so on. As such, only a small fraction of the computation time per image is dedicated to the simulation. Furthermore, as user inputs might be unpredictable, stability of the simulation is of uttermost importance. Because of these constraints, physical correctness in real-time simulations is even more difficult to achieve than for their offline counterparts. Physical equations are often simplified, sometimes using bold assumptions, and even frequently approximated with a non-physically based model in order to improve computation times. Furthermore, some constraints such as compressibility and forces, e.g., gravity, are also often relaxed. In both the offline and real-time cases, compromises in terms of visual details of the simulation have to be made in order to meet both the computations time and memory restrictions.

Visual details of a liquid can be seen through its motion and the size of its surface features. A highly detailed liquid simulation captures well the motion and shape of small-scale droplets and thin sheets of liquids. However, doing so requires a very large amount of simulation data, e.g., 3D particles and/or cells in a volumetric grid. In some cases, a significantly smaller amount of simulation data may be required to capture well its motion than is required to capture its

finer surface features. For example, a melting plastic toy may slowly deform without generating small splashes or thin sheets. However, its surface may contain finer details that we wish to retain as the simulation progresses. Unfortunately, the resolution of the simulation, which directs the motion, is generally highly related to the resolution of the surface because the surface is either encoded within the simulation data, e.g., a signed distance field, or built from the data, e.g., implicit function computed from the position and radius of simulation particles. As such, it is not possible to lower the amount of simulation data without also affecting the size of the features captured by the surface. To solve this issue, previous work use an explicit mesh surface (Wojtan, Thürey, Gross & Turk, 2009, 2010) whose representation is independent of the simulation data. This mesh surface is generally provided by the user to the solver, which then updates it throughout the simulation to follow the motion of the liquid. This works well for structured simulation data such as a grid or tetrahedral mesh because they can be adapted to track the interior of the liquid using the explicit mesh surface. However, this is not as trivial with particle-based methods, where a drift between the explicit mesh surface and the particles is often seen. To fix this issue, the explicit mesh surface can be projected on the surface on an implicit surface built from the particles (Yu, Wojtan, Turk & Yap, 2012). However, this projection does not preserve the finer details of the explicit mesh surface. In chapter 2, we address this limitation and describe an approach that preserves the finer geometrical features of a high resolution 3D mesh surface using a detail-preserving projection, making it possible to significantly lower the resolution of the underlying particle-based simulation while preserving the finer features of its surface.

Some granular materials, although different from fluids, can be simulated using similar paradigms. Particle-based approaches such as the smoothed-particle hydrodynamics (SPH) method (Ihmsen, Wahl & Teschner, 2013) and the unified particles framework described by Macklin, Müller, Chentanez & Kim (2014) have been successfully used to simulate sand. Hybrid approaches, such as the fluid implicit particle method (FLIP) (Zhu & Bridson, 2005) and the material

point method (MPM) (Klár *et al.*, 2016), have also shown very good results. The latter has been used to simulate snow as well (Stomakhin, Schroeder, Chai, Teran & Selle, 2013). In all of these approaches, the granular material is represented by small particles. Because the size of sand grains and snowflakes is very small in real life, it is often not possible for the simulation particles to match the size of the real-life counterpart, whether it is because of time or memory constraints. This limitation becomes greater as the volume of sand or snow becomes larger, e.g., a vast and thick layer of snow on the ground. In such cases, it is not uncommon to encounter scenarios where only a small fraction of the particles are actually moving. Consider, for example, an animated character walking in the snow. While the simulation domain may contain a large volume of snow, its particles only need to be update when they interact directly or indirectly with the character. Some work handles such cases by deactivating particles that come to rest, making it possible to simulate much larger simulation domains in real life VFX scenarios (Klár, Budsberg, Titus, Jones & Museth, 2017) by focusing computations only where needed. While such optimization may considerably improve computation times, memory may become a bottleneck which limits the volume of material or the resolution of the particles in the simulation. In chapter 3, we describe an efficient workflow for simulating a layer of snow on the ground, which focuses memory and computational times only in regions that interact with the animated characters or objects. This workflow uses a lightweight volumetric representation for snow that is static, and transfers parts of its volume to a particle-based simulation as it interacts with animated characters and objects.

As stated previously, the time allocated to running physical simulations in real-time applications is very small. This imposes a very large constraint on the number of elements that can be simulated. To lower the computation times required to run, for example, a liquid simulation, it is often processed on the GPU. This further imposes a memory constraint because graphics cards have access to much less memory than the CPU normally does, especially when you consider all the other resources it must coexist with, e.g., 3D models, textures, etc. The combination of these

time and memory constraints means that the resolution of a full 3D simulation may be very low in practice. For liquids, this considerably limits the ability to correctly capture smaller features such as tiny droplets from splashes and thin sheets that form as they flow on a surface. Previous work rely on a 2D height map grid to simulate liquids efficiently (Chentanez & Müller, 2010; Borgeat, Massicotte, Poirier & Godin, 2011; Kellomäki, 2014). Each cell is a column of liquid whose height varies as the liquid flows, which makes it possible to simulate 3D volumes of liquids with much less data. Such techniques cannot recreate complex 3D behaviors such as splashes with sufficient fidelity, but they work well for volumes of liquids that flow and have ripples, e.g., a river, a lake, etc. Nonetheless, they can simulate very thin sheets of liquids, which can make them ideal for simulations where a small layer of liquid flows on a surface. These techniques are generally limited to a single continuous surface and cannot handle surfaces with overhangs very effectively. In chapter 4, we describe an approach for real-time liquid simulation with a focus on real-time blood simulation for surgical VR training. It extends previous approaches that rely on 2D heightmap by proposing an improved surface reconstruction technique that better handles surfaces with overhangs. Furthermore, it introduces a new physically-based viscosity model that is both stable and efficient. This new model aims to better recreate the effect of the friction between the underlying material and the column of viscous liquids such as blood.

All of the contributions above are on the common theme of efficient fluid simulation. They aim to generate high quality simulations that exhibit finer visual details with smaller computation times and memory requirements. For the VFX industry, which our first two approaches target, this allows artists to iterate faster toward a desired artistic vision, and also to run simulations at a resolution that would normally not be possible because of memory constraints. For real-time simulations, as targeted by our last approach, it allows higher-quality simulations without exceeding the very limited time budget.

Both the explicit mesh surface and the real-time liquid simulation approaches were validated and compared against state of the art methods, while the efficient workflow for simulating snow did not have comparable state of the art methods. These contributions lead to three journal and one conference papers.

CHAPTER 1

LITERATURE REVIEW

This chapter contains an overview of the previous work related to this thesis. A general introduction on fluid simulation techniques for computer graphics is first provided, before discussing more precise work on surface tracking, snow/granular simulation using fluid simulation paradigms, and real-time fluid simulation. Note that while both liquids and gases are fluids, this thesis focuses more on materials with a liquid-like behavior, such as water and blood, as well as materials that exhibit both solid and liquid behaviors, e.g., melting wax, snow, etc.

1.1 Overview of Fluid Simulation in Computer Graphics

Fluid simulation techniques are generally divided into two categories, Eulerian or Lagrangian, which have two distinct ways of modeling the fluid motion. Eulerian techniques observe the fluid at fixed points in space and analyze how its properties, e.g., velocity, density, pressure, etc., evolve over time. On the other hand, Lagrangian simulations follow the individual fluid elements whose positions are changing over time.

In the field of computer graphics, purely Eulerian simulations (Foster & Metaxas, 1996; Enright, Marschner & Fedkiw, 2002) generally rely on a uniform grid structure to evaluate the fluid properties and velocities at the center of the cells or at the center of their shared faces. To account for the fluid movement through the grid, these properties need to be updated by solving an advection term. To guarantee stability with smaller computation times, the semi-Lagrangian advection (Stam, 1999) handles this term by tracing back the original fluid position to handle this term. However, this method is prone to numerical dissipation which smooths the fluid properties, including the velocity. While some work lower this effect (Selle, Fedkiw, Kim, Liu & Rossignac, 2008), the size of the velocity field features that can be preserved is generally limited by the size of the grid cells. This is also true for other properties such as the density or surface information.

As for the Lagrangian techniques, particle-based methods using the SPH paradigm are among the most popular in the computer graphics literature. These methods update the particles' velocities

and positions from forces computed from particles in the neighborhood of each particle. Early work integrated explicitly the particles' velocity and position (Desbrun, Cani *et al.*, 1996; Müller, Charypar & Gross, 2003; Becker & Teschner, 2007), which required a small timestep to preserve stability. However, more recent work relies on implicit integration (Ihmsen, Cornelis, Solenthaler, Horvath & Teschner, 2014a; Bender & Koschier, 2017) or constraints minimization (Macklin *et al.*, 2014) to improve stability with larger timesteps. These particle-based methods have the advantage of better preserving the fluid's volume over time compared to their Eulerian or hybrid counterparts. Furthermore, they scale well to irregular simulation domains that would normally contain a lot of empty cells using a traditional grid. Also, it is much easier for users to work with particles than with volumes, which makes these approaches, as well as hybrid approaches, more artist friendly than purely Eulerian simulations. However, they can be much more computationally expensive because the neighborhood of a particle must be recomputed every timestep, and a single particle generally interacts with a much larger neighborhood (~30-80 particles) than a grid cell (6 cells, in 3D) does.

Most current fluid solvers in computer graphics rely on a hybrid approach such as the FLIP and the Particle-in-cell (PIC) methods (Zhu & Bridson, 2005), the affine-particle-in-cell (APIC) method (Jiang, Schroeder, Selle, Teran & Stomakhin, 2015), the Power Particle-in-Cell method (Qu, Li, De Goes & Jiang, 2022), or the MPM (Stomakhin *et al.*, 2014; Su, Xue, Han, Jiang & Aanjaneya, 2021). In these approaches, particles are used to advect the fluid quantities efficiently, while other calculations, such as viscosity handling and pressure projection, are solved more efficiently on the grid. Such approaches are generally less prone to numerical dissipation while being more efficient than a particle-based simulation with the same number of particles. Furthermore, the particles can capture visual features that are finer than the cells size of the underlying grid. Note that with hybrid techniques, the particles do not interact directly with each other. To lower the computation times and memory requirements of FLIP simulations, some work use FLIP particles only within a narrow band (Ferstl, Ando, Wojtan, Westermann & Thuerey, 2016; Sato, Wojtan, Thuerey, Igarashi & Ando, 2018). Alternatively, some approaches use a spatially adaptive simulation (Ando, Thürey & Wojtan,

2013; Ando & Batty, 2020) to focus computations where they are needed the most. Some work instead focuses on augmenting a coarser simulation (Roy, Paquette & Poulin, 2020) by adding more particles around splashes as a post-process step to introduce finer visual details with lower computation times. They also introduce an implicit wave model that propagates the impact of the newly added particles to the coarser simulation. To increase the apparent resolution of a coarse simulation, Roy, Poulin & Paquette (2021) correct the particles motion using a deep learning approach to match the behavior of a much higher resolution simulation.

1.2 Surface Tracking

While in most simulations the forces between air and the liquid are neglected, the interface between these two fluids remains an important aspect of liquid simulation. The process of determining where the surface is, and how it evolves throughout the simulation is called surface tracking. Knowing where the surface of the liquid lies is important during visualization. Furthermore, it is also required by Eulerian simulations to determine which cells are part of the liquid and which are not, which makes it vital to the accuracy of the simulation, and may result in volume gain/loss if not properly handled.

A wide range of work focused on tracking the surface of liquid simulations. The various techniques can be split into two main categories depending on their underlying surface representation: implicit or explicit. With implicit techniques, the surface is expressed as an isosurface of a 3D function, which can then be triangulated for visualization. For explicit techniques, the surface is explicitly expressed as a polygonal mesh that is evolved and re-meshed through time.

1.2.1 Implicit Surface Tracking

The most common fluid surfacing techniques build an implicit representation of the surface encoded into a grid-like structure. To visualize the surface, a triangle mesh is generated at each frame from the implicit representation using a method such as marching cubes (Lorensen & Cline, 1987).

A common way to represent the surface of a liquid with an Eulerian simulation involves using a level set. Foster & Fedkiw (2001) introduced a level set method that uses particles, where both the level set and the particles are advected using the fluid velocity field. Particles are used to correct the level set after each advection stage, thus reducing volume loss. Later, this method was improved by Enright *et al.* (2002) with particles on both sides of the interface, further improving volume preservation. This method has later been adapted to support adaptive grids (Losasso, Gibou & Fedkiw, 2004) and grids with finer resolutions than the underlying simulation (Kim, Song & Ko, 2009), making it possible to create a more detailed surface using a coarse simulation. It is worth mentioning that the surface representation is also used by Eulerian approaches to track cells inside the liquid and apply boundary conditions near the free surface during the simulation. Thus, a high-quality surface has a great impact on both the simulation and its visualization.

For particle-based simulations, most approaches encode the surface inside a grid as an implicit function that is computed from the particles. Müller *et al.* (2003) triangulate the isosurface of a color field computed using the SPH framework to create a surface from liquid particles. Even though the surface field is smoothly interpolated between particles, the resulting surface is prone to bumpiness. Later, Zhu & Bridson (2005) developed a method based on the distance to the average position of particles within an influence radius. This method provides a smoother surface, but is prone to generating erroneous volumes of fluids between splashes and in concave regions. This problem was resolved by Solenthaler, Schläfli & Pajarola (2007) with the use of an attenuation factor in those regions. Adams, Pauly, Keiser & Guibas (2007) also improved the technique of Zhu and Bridson by storing a signed distance field at each particle's position, which allowed the generation of a smoother surface with irregular particle distribution and with particles of varying radii. However, updating the sign distance field makes it more computationally intensive than previous approaches. Yu & Turk (2013) used anisotropic kernels to provide a smooth surface. Their method generates a smooth surface while being faster than that of Adams *et al.* (2007). However, it involves longer computation times than the method of Zhu and Bridson, and suffers from volume shrinkage because the particles' positions are smoothed

beforehand. In their adaptive liquid simulation framework, Ando *et al.* (2013) used the union of convex hulls built from groups of three particles to generate a smoother surface, at the expense of longer computation times. Other techniques focus on first generating a lower quality surface, and smoothing it afterward. The smoothing operation is either done on the triangulated mesh (Williams, 2008), or on the implicit function (Bhatacharya, Gao & Bargteil, 2011). In contrast to Eulerian simulations, the surface representation is not needed by particle-based approaches during the simulation, allowing a more flexible workflow where the surface can be parameterized and built after the desired liquid behavior is obtained.

All these methods (Adams *et al.*, 2007; Ando *et al.*, 2013; Bhatacharya *et al.*, 2011; Müller *et al.*, 2003; Solenthaler *et al.*, 2007; Williams, 2008; Yu & Turk, 2013; Zhu & Bridson, 2005) generate a new surface from the simulation particles for every frame of the animation. Therefore, they depend on the resolution of the underlying simulation to provide a detailed surface. As such, they are not very accurate in preserving the features from the initial surface used to generate the simulation particles. This can be particularly problematic when a highly detailed surface is needed, while a coarse simulation can effectively capture the desired behavior. A good example would be the simulation of a melting object, where the initial surface is a detailed user-supplied mesh.

1.2.2 Explicit Surface Tracking

In order to preserve the finer details of a user-provided surface, other methods update an explicit mesh surface throughout the simulation. Brochu & Bridson (2009) developed a framework for tracking explicit surfaces where triangle collisions are computed, and mesh surgery is performed directly on the mesh triangles to handle splitting and merging of the surface. Mesh improvement operations, such as edge collapse and edge splitting, are also performed on the surface to guarantee its quality. Their framework has successfully been used to model the surface of liquids with thin features (Edwards & Bridson, 2014), and has been extended to handle multimaterial interfaces (Da, Batty & Grinspun, 2014). Müller (2009) advects an explicit mesh using the velocity field of an Eulerian simulation. Intersections between the mesh and cells from a 3D

grid are then computed and used to rebuild the surface entirely using marching cubes templates. They also indtroduce new template configurations to support thin sheets of liquid. However, other small-scale surface details, such as ripples, are not preserved. Wojtan *et al.* (2009) developed a hybrid mesh grid method in which merging and splitting regions are identified by comparing the mesh with its implicit representation, which is a signed distance function stored on a grid. These regions are then remeshed locally from the implicit representation. This effectively preserves features of the explicit mesh in regions where no topological changes have occurred. Furthermore, the quality of the topology change detection and the remeshing is only dependent on the implicit representation grid resolution. Thus, the simulation resolution can be coarser while preserving a detailed surface. The same authors (Wojtan *et al.*, 2010) improved this work by maintaining sheets of liquids thinner than the grid resolution. Although the grid resolution still affects the quality of the surface, coarser grids can be used than with their previous work. More recently, Heiss-Synak *et al.* (2024) generalized hybrid mesh-grid approaches to non-manifold surface meshes to track the surface of multi-material flows. These techniques have been used with Eulerian fluid simulations (Müller, 2009; Wojtan *et al.*, 2009, 2010), as well as with Lagrangian finite element methods (FEM) (Wojtan *et al.*, 2009) and the discontinuous Galerkin method (Edwards & Bridson, 2014). In all cases, the explicit mesh is used by the fluid simulation solver to track the fluid's interior.

Yu *et al.* (2012) adapted the work of Wojtan *et al.* (2009) for particle-based simulations. Because particles, instead of the explicit mesh, are used by the solver to track the fluid's interior, a divergence is often seen between the explicit surface and the particles during mesh advection. To prevent this problem, the mesh is projected onto an implicit surface representation every frame. While this projection stage does not preserve surface details, this is however not a problem in their case since their goal is to use an already coarse mesh to track surface properties over time. This projection was also used by Chentanez, Müller, Macklin & Kim (2015) with their fast grid-free explicit mesh surface tracking method, which also suffers from the same loss of surface details. While our approach described in chapter 2 also projects the mesh vertices on an isosurface of the particles' implicit function, the projection of Yu et al., based on the mesh

vertices' normals, is less precise than ours, and does not preserve the explicit mesh surface details.

Explicit surface representations used with Eulerian or FEM simulations have the advantage of being able to preserve surface details without the need for a finer simulation resolution, but the fluid simulation and explicit surface advection are interlinked, forcing the simultaneous calculation of both. On the other hand, with particle-based simulations, it is possible to rebuild a new surface using modified surface reconstruction parameters without having to run the simulation all over. However, to preserve the consistency of the explicit surface with the simulation particles, current techniques project the explicit surface onto a coarser surface built from the particles, which then loses high-resolution surface details. The approach proposed in chapter 2 focuses on preserving these surface details, while enforcing the consistency of the explicit surface with the simulation particles.

1.3 Real-Time Fluid Simulation of Shallow Liquids

In the field of computer graphics, most fluid animation efforts concentrate on offline simulations using either an Eulerian (Enright *et al.*, 2002), particle-based (Ihmsen, Orthmann, Solenthaler, Kolb & Teschner, 2014b; Bender & Koschier, 2017), or hybrid (Jiang *et al.*, 2015; Su *et al.*, 2021; Qu *et al.*, 2022) simulation. Some work focuses computation on areas with more details using adaptive grid structures (Aanjaneya, Gao, Liu, Batty & Sifakis, 2017; Ando & Batty, 2020), narrow band surfaces (Ferstl *et al.*, 2016; Sato *et al.*, 2018), or adaptive particle radii (Winchenbach, Hochstetter & Kolb, 2017; Winchenbach & Kolb, 2021). These methods are capable of generating astonishing visual results, but are unfortunately too slow for real-time applications. Macklin & Müller (2013) were able to achieve impressive visual results by simulating and rendering more than $100,000$ particles in real time using their position-based dynamics framework. While this method is real-time, its ability to reproduce smooth thin layers of liquid is limited.

To reduce memory and computational costs, it is more efficient to use a heightmap to represent the fluid and perform a 2D simulation to update the liquid's height. While methods using a heightmap cannot exhibit some more complex behaviors such as splashes and wave crests, they are still adequate for a broad range of scenarios. For example, intricate wave patterns in shallow water can be encoded as height displacements by particles (Yuksel, House & Keyser, 2007) or packets of similar wavelengths (Jeschke & Wojtan, 2017). Furthermore, heightmap methods can simulate arbitrarily thin films of liquids with no impact on the simulation resolution. Lee & O'Sullivan (2007) allowed some compressibility in their 2D simulation based on the Navier-Stokes equations and adjusted the liquid's height based on its density. While simple and efficient, this technique does not account for the underlying terrain elevation. By assuming a vertical anisotropy of the liquid's velocity, the Navier-Stokes equations can be simplified, resulting in the shallow water equations, which can be further simplified to the shallow wave equations (Kass & Miller, 1990). Several papers focus on implicitly solving these equations on a regular grid (Kass & Miller, 1990; Layton & van de Panne, 2002) or on triangular mesh surfaces (Wang, Miller & Turk, 2007; Angst, Thuerey, Botsch & Gross, 2008). Methods with an implicit integration maintain stability at larger timesteps, but are prone to a lot of diffusion as well as volume gain when using a large timestep. Furthermore, faster-moving boundaries require a smaller timestep, which can considerably increase the computation time. On the other hand, Chentanez & Müller (2010) showed that their explicit integration of the shallow water equations (SWE) is able to simulate large-scale scenarios in real time. For their part, our experiments show that the explicit integration requires a considerably smaller timestep for small-scale examples because of the larger ratio between the liquid's velocity and the simulation cell size, which limits its use for real-time applications in that context.

A simpler model for simulating shallow water, the virtual pipes (VP) method, was introduced by O'Brien & Hodgins (1995). It is based on the hydrostatic pressure difference between neighbor cells of a uniform grid. Liquid is transferred between them through virtual pipes connected at their bottom, and the simulation uses an explicit integration. This method has been extended to support multi-layered heightmaps in order to allow simulations above partially submerged

floating obstacles (Kellomäki, 2014) and on more complex terrains with overhangs (Borgeat *et al.*, 2011). Furthermore, our experiments show that the VP method allows a considerably larger timestep than with the shallow water method of Chentanez & Müller (2010) for small-scale scenarios. As such, the approach described in chapter 4 builds upon the VP method to simulate real-time, small-scale shallow waters.

Because, the surface construction of previous methods does not handle the changing topology of multi-layered simulation surface well, or not at all, we improve the surface to account for its evolving multi-layer topology. Furthermore, prior VP and SWE methods often ignore effects that are visible at a smaller scale, such as the viscous drag force from the terrain. Our experiments also showed that solving the viscosity term directly on the grid is either unstable in small-scale scenarios using an explicit integration, or computationally expensive using an implicit integration. To approximate the viscosity from the ground to the surface of the liquid, one could use a drag force such as the one described by Mould & Yang (1997). While this technique would be both very fast to compute and stable, it is not physically-based and does not account for the amount of liquid in a column, which affects the viscous behavior of the liquid. In chapter 4, we thus derive a physically-based viscosity model. For faster computation and stability, we neglect neighbor contributions and focus on the friction from the ground to the surface of the liquid. Our resulting model is similar to a variable drag coefficient, but accounts for the liquid's height and is physically based.

1.4 Snow Simulation

Early work on snow focused mostly on modeling its accumulation on the ground and objects. Some methods analyzed the flow of snow (Fearing, 2000; Moeslund, Madsen, Aagaard & Lerche, 2005) as it falls from the sky, while others relied on geometric models (Festenberg & Gumhold, 2009, 2011) to generate the snow in a static scene. Some work used physical simulations to model the impact of wind (Feldman & O'Brien, 2002; Wang, Wang, Xia & Peng, 2006) and heat transfer (Maréchal, Guérin, Galin, Mérillou & Mérillou, 2010). Nevertheless, these methods

focus mainly on snow accumulation. Thus, they do not address the animation of snow that interacts with dynamic objects.

It is quite common to consider snow as a granular material, like sand. Sumner, O'Brien & Hodgins (1999) used a height map to animate sand, mud, as well as snow. In their method, the snow height map is updated when a collision occurs with a dynamic object. Cells surrounding the colliding cells are then updated to take into account mass displacement and erosion. Particles are created at the surface of the colliding objects to model snow that adheres to these objects and then falls. As particles fall back to the ground, volume is added to the corresponding height map cells. This method was improved by Onoue & Nishita (2005), by allowing snow to lie on top of objects through the use of a multivalued height map. Zeng *et al.* (2007) further improved the behavior by considering the velocity of the dynamic objects when computing the mass displacement. Pla-Castells, García-Fernández, Martinez-Dura *et al.* (2008) used cellular automata to compute the mass displacement between height map cells in a more physically correct way. Inspired by the shallow water equations, Zhu, He, Li, Wang & Wang (2021) proposed the shallow sand equations (SSE) to simulate 2D granular materials. They use a two-layer model which does not restrict the depth of the material. Their work was later improved by Su *et al.* (2023) by adding an internal elasto-plastic force. These approaches were designed for real-time applications and are fast to compute, however, they are very limited in the variety of behaviors they can reproduce. The height map behavior is limited to a 2D simulation, and cannot model the dynamics of airborne snow. While some of these methods (Onoue & Nishita, 2005; Sumner *et al.*, 1999) use particles to depict airborne sand or snow, their dynamics is very limited. Inter-particle interactions are not considered, and particles do not interact with dynamic objects after being emitted. Furthermore, the transfer of mass from particles to height map cells will likely produce temporal artifacts in the animation.

Fully physically-based simulations handling the interaction between particles and dynamic objects enable the complete range of snow dynamics. Bell, Yu & Mucha (2005) used the discrete element method (DEM) to simulate sand using particles. The long computation times of their method were shortened significantly in the work of Zhu & Yang (2010) by converting

still particles hidden under surface particles to a multivalued height map, and only simulating the surface particles. A variety of paradigms have also successfully been adapted to simulate granular materials, such as FLIP (Narain, Golas & Lin, 2010; Zhu & Bridson, 2005) and SPH (Alduán & Otaduy, 2011; Ihmsen *et al.*, 2013; Lenaerts & Dutré, 2009). Additionally, position based dynamics (Müller, Heidelberger, Hennix & Ratcliff, 2007) have been used by Macklin *et al.* (2014) to simulate sand using collision and friction constraints. Recently, Yu, Li, Lan, Yang & Jiang (2024) introduced eXtended Position-Based Inelasticy (XPBI), which better simulate granular materials, such as sand and snow, with a model based on their standard constitutive laws. While these simulation paradigms enable a rich range of dynamic behaviors, they do not model some of the characteristics specific to snow, such as its compressibility and its interaction with air.

Instead of simulating any granular material, some fully physically-based simulation methods specifically address the simulation of snow. Takahashi & Fujishiro (2012) extended the SPH method to handle the sintering of snow, i.e., snow that compacts under pressure or heat. Stomakhin *et al.* (2013) used MPM to simulate wet and dense snow using an elasto-plastic model. Their hybrid Eulerian/Lagrangian approach allowed them to achieve a visually plausible simulation for snow that exhibits both fluid and solid properties. Later, Gissler, Henne, Band, Peer & Teschner (2020) used a similar model in their implicit SPH simulation of snow. They split the compressible pressure and shear parts of the elastic deformation into two different solvers, which makes it possible to couple their approach with other particle-based simulations. Wong, Fu *et al.* (2015) simulated snow using a DEM simulation with cohesive forces. Similarly to Zhu & Yang (2010), they reduce computation times by only considering surface particles. More recently, Andreasson, Östergaard & Goswami (2024) proposed a spatial and temporal adaptive DEM snow simulation that showed performance improvements in simulations with a large number of particles. While these simulation methods can model a wide range of snow behaviors, their computation time and memory requirements become unmanageable when considering larger scenes or when requiring finer details. Thus, they generally scale poorly in scenes that cover a large area. On the other hand, as stated previously, methods based on multivalued height

maps are limited in the range of behaviors that they can model, but are much faster and scale better to larger scenes.

As with multivalued height map methods, the approach described in chapter 3 has low computation time and memory costs. Nevertheless, we rely on a level set instead of a grid, resulting in more flexibility and precision for the dynamic object interactions. The level set also allows us to achieve a higher degree of realism by applying a 3D localised noise where imprints appear in the snow. Additionally, to allow for the wider range of behaviors of the fully physically-based methods, the approach proposes an adapted and localized granular simulation. Furthermore, the computation times of our simulation are lowered by computing its dynamics only where potential interactions may occur. Finally, to reproduce the full spectrum of snow behavior, our snow mist uses a modified fluid simulation to model the interaction with the surrounding air. Thus, our approach allows more visually complex results than multivalued height map methods, while requiring less memory and computation times than a full particle simulation.

CHAPTER 2

FLUID SIMULATION WITH SURFACE TRACKING

The work described in this chapter has been published in the *Computer Graphics Forum* journal (Dagenais, Gagnon & Paquette, 2017):

François Dagenais, Jonathan Gagnon, and Eric Paquette. *Detail-Preserving Explicit Mesh Projection and Topology Matching for Particle-Based Fluid*, Computer Graphics Forum, Volume 36, Number 8, pp 444–457, 2017.

Particles are commonly used in fluid simulation, particularly with the rise in popularity of the FLIP (Zhu & Bridson, 2005) and APIC (Jiang *et al.*, 2015) methods found in commercial software (such as Houdini™, Maya®, and Realflow™). When simulating liquids and reconstructing the surface from the particles, the irregular distribution of particles, as well as the spherical nature of the implicit functions typically used for reconstruction, make the resulting surface prone to bumpiness. This is shown in Fig. 2.1, where the Stanford Armadillo is filled with different number of particles and then its surface is reconstructed from the particles. As can be seen, a coarser simulation will severely smooth the details; therefore, a large amount of particles is needed to accurately reconstruct the details of the original mesh. Even in Fig. 2.1d, where over four million particles were used, some details around the eyes and the teeth have been smoothed out. Consider, for example, the case of a melting plastic toy. This effect can be simulated using a particle-based phase change simulation (Dagenais, Gagnon & Paquette, 2012; Stomakhin *et al.*, 2014). In this scenario, it is important to retain the details of the original mesh where its surface hasn't melted yet, and this requires a large number of particles. However, such a detailed simulation might be unnecessary, considering the small amount of detail present in the fluid movement. This makes the simulation and surface reconstruction times unnecessarily longer, whereas the objective is only to improve the reconstructed surface.

Explicit surface tracking (Wojtan, Müller-Fischer & Brochu, 2011) helps to solve this problem by using an initial mesh and evolving it based on the underlying simulation. Remeshing is done throughout the process in order to preserve the coherence and the quality of the surface.

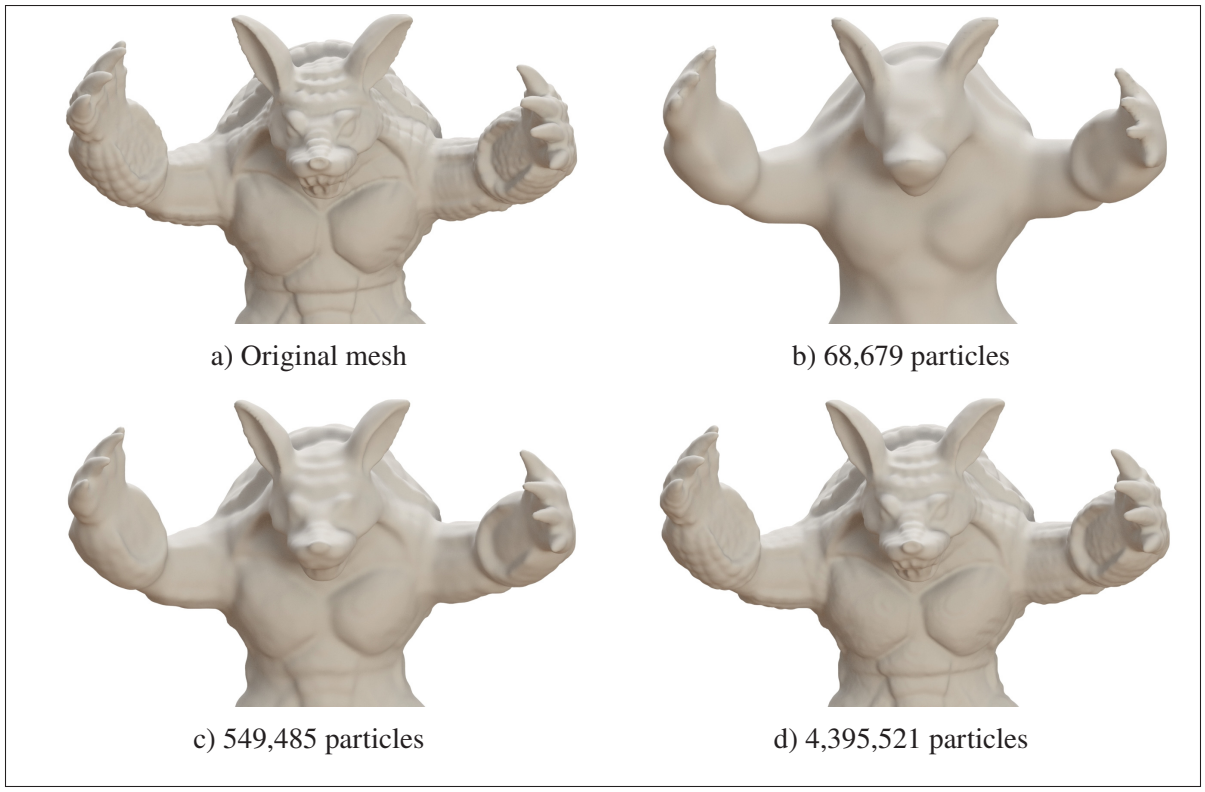


Figure 2.1 Comparison between the original surface and reconstructions with varying resolutions. The surfaces were generated using the method of Solenthaler *et al.* (2007)

However, explicit surface tracking is not well suited for particle-based simulations since the explicit mesh tends to diverge from the particles. While recent work addresses this issue (Yu *et al.*, 2012), it does not preserve surface details very well. Since it relies on a projection onto a surface constructed from the particles, the size of the explicit surface's details is still limited by the resolution of the underlying simulation.

Our approach focuses on improving and extending explicit surface tracking approaches for particle-based fluids in order to retain surface details while maintaining a behavior consistent with the simulation particles. It does so by extracting high-resolution details based on the distance between an initial surface mesh and a coarse implicit surface representation built from the particles (see Fig. 2.2). This distance is then preserved by our improved projection. Furthermore, we introduce a novel topology matching operation that preserves the consistency of the explicit surface with the behavior of the particles. Altogether, this allows the tracking of a

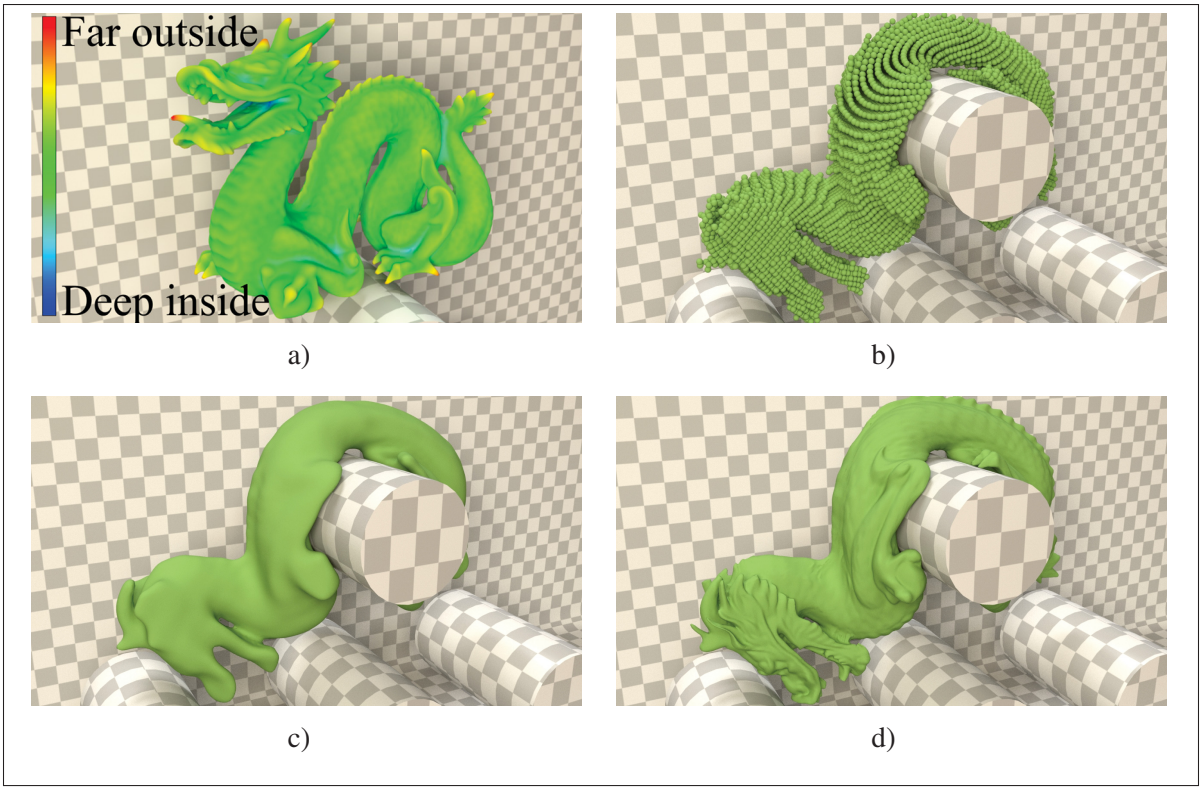


Figure 2.2 High-resolution details are extracted based on the distance between an initial surface mesh and its coarse implicit surface representation (a). We use the particles' behavior and the topology of a coarse resolution simulation (b) to consistently evolve an explicit surface mesh (d). The high-resolution details are lost if using only the implicit surface from the coarse set of particles (c), but our explicit mesh surface approach preserves surface details (d), even with such a coarse resolution fluid simulation. In (a), the surface is color-coded based on the distance from the implicit surface, where blue refers to the largest inward distances, red refers to the largest outward distances, and green is at the isosurface. The coarse simulation (b) contains only 20,456 particles

detailed explicit mesh surface by using a coarser particle simulation. Moreover, the approach can be run entirely as a post-simulation step, and it is independent of the simulation method, having been used with both FLIP and SPH simulations. Our main contributions can be summarized as follows:

- A new explicit mesh surface tracking technique for particle-based simulations.
- A detail-preserving projection based on a signed distance field.

- A novel topology matching approach that reconstructs a section of a mesh implicit representation based on the topology of a reference signed distance field.

2.1 Overview of the Detail-Preserving Surface Projection Approach

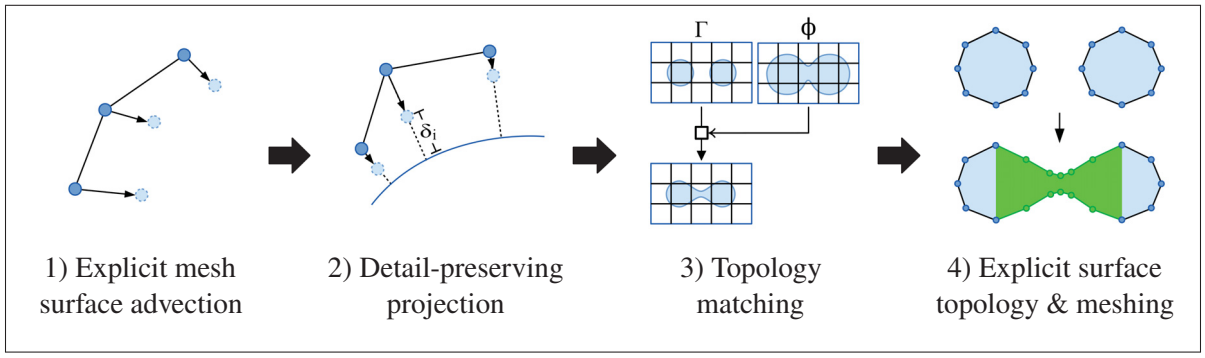


Figure 2.3 Overview of our four-step approach

We introduce a new detail-preserving surface projection approach that extends the work of Yu *et al.* (2012). Using our projection, combined with a novel topology matching operation, changes in the topology of a coarse implicit surface are reflected onto a detailed explicit mesh surface. The global approach (see Fig. 2.3) consists of four main stages:

1. Advection (Yu *et al.*, 2012) (Sec. 2.2)
2. New detail-preserving projection (Sec. 2.3)
3. New topology matching (Sec. 2.4)
4. Explicit surface topology & meshing (Wojtan *et al.*, 2009) (Sec. 2.5)

During the first stage, the positions of the explicit mesh vertices are updated using the positional changes of the underlying simulation particles. Afterward, the explicit mesh is projected (stage 2) using the fluid implicit function, while preserving the mesh features. After the mesh has been advected and projected, an implicit representation of the explicit mesh is computed and modified to match the topological changes of the implicit surface (stage 3). Finally, the modified mesh implicit representation is used to locally rebuild the explicit mesh surface (stage 4), only where needed, based on the topological changes (identified in stages 2 and 4).

2.1.1 Data Structures

The approach presented in this chapter maintains an explicit mesh structure and three scalar grid data structures. The three grid structures are the mesh implicit representation Γ (see Sec. 2.4), the discretized implicit function ϕ (see Sec. 2.3.1), and the difference field ψ (see Sec. 2.4). A grid cell can be accessed either with its 1D index (e.g., ϕ_i) in memory, its 3D index (e.g., $\phi_{i,j,k}$) in space, or a vector (e.g., $\phi(\mathbf{x}_i)$) denoting a position in space. The grid resolutions are based on r, the initial distance between simulation particles. To preserve temporal coherence, the cell size is kept fixed, and the location of cells is kept fixed along the x, y, and z axes. To adapt to the changing fluid, only the extent of the grids is updated at each frame to fully surround the explicit mesh and the particles.

2.2 Explicit Mesh Advection

In order to advect the explicit mesh, the method of Yu *et al.* (2012) uses the particle velocities to update the vertex positions at every timestep, i.e., iteration, of the simulation. To preserve stability of the simulation and preserve a more accurate behavior, the simulation timestep is often smaller than the time increment between two frames of the animation. This means that in their approach the advection is performed several times per frame. In our approach, the advection is instead done solely at each frame, and relies on particle position updates. Using the position update instead of the particle velocity allows the surface to be advected only once per animation frame, instead of once per simulation timestep, thus reducing the computational expense. As there are multiple timesteps per animation frame, this eliminates the need to store particle data after every timestep in order to update the surface after the simulation is run. The particle position update $\Delta\mathbf{p}_j$ is used as follows:

$$\mathbf{x}_i^{t+\Delta t} = \mathbf{x}_i^t + \frac{\sum_j \Delta\mathbf{p}_j W_{ij}}{\sum_j W_{ij}},$$

where W_{ij} is the weighting function and $\mathbf{x}_i$ is the vertex position. In our explicit mesh surface examples, the *poly6* weighting function from Müller *et al.* (2003) is used. The influence radius

is set to $2r$, where r is the initial distance between the simulation particles. When a vertex has no neighbor within the influence radius, the radius is expanded until at least one particle is found. This guarantees that every vertex of the explicit mesh is updated.

2.3 Detail-Preserving Projection

The advection stage described earlier is prone to small errors that accumulate during the course of the simulation. This results in a surface that diverges slowly from the particles, and in small bumps that tend to gradually appear at the surface of the mesh (see Fig. 2.4a). Furthermore, two surfaces moving toward each other do not always merge as they should. This problem is depicted in Fig. 2.4a, where the front paws of the bunny did not merge with its chest, as would have been expected. This is also true with splitting (see Fig. 2.4c), and results in visually unpleasant stretching and volume gain. As in the work of Yu *et al.* (2012), the explicit mesh is projected onto the isosurface of an implicit function. While Yu et al. projected along the vertex normals of the explicit mesh, we project using the gradient of the implicit function, which is a signed distance field. Projecting along this gradient is much more precise, as with our detailed surfaces, the normal might not be in the direction of the shortest path to the surface, and may not even cross the surface (see Fig. 2.5). Moreover, it might cross the projection path of another vertex, resulting in a surface with flipped triangles.

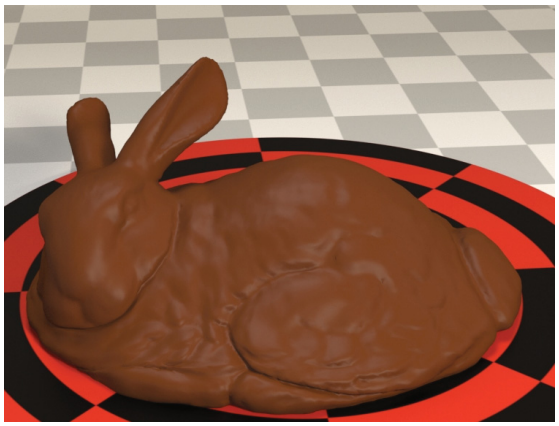
The projection is based solely on the signed distance field, and details are preserved by maintaining the same distance between a vertex of the explicit mesh and its projection for each frame of the animation. This is done in two steps (see Fig. 2.6):

1. Construction of the implicit function ϕ (Fig. 2.6a & Sec. 2.3.1)
2. Projection of the explicit mesh (Fig. 2.6b & Sec. 2.3.2)

2.3.1 Implicit Function

We chose our implicit function ϕ to be a signed distance field. This guarantees that, for any vertex position $\mathbf{x}_i$, the gradient $\nabla\phi(\mathbf{x}_i)$ will be in the direction away or toward the nearest point

a) Without projection: bumps and unhandled merges



b) With projection

c) Without projection: unrealistic stretching and volume gain

d) With projection

Figure 2.4 Examples without (left) and with (right) our detail-preserving projection

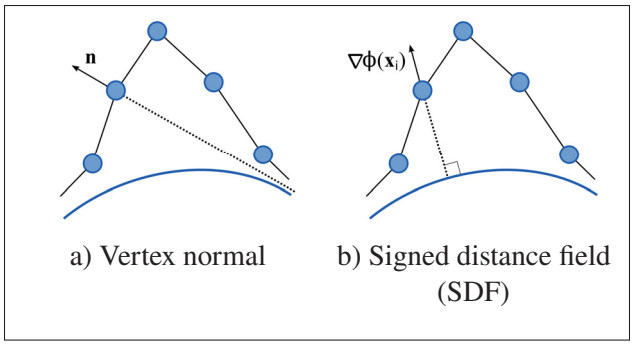


Figure 2.5 Projection using the vertex normal vs. a signed distance field

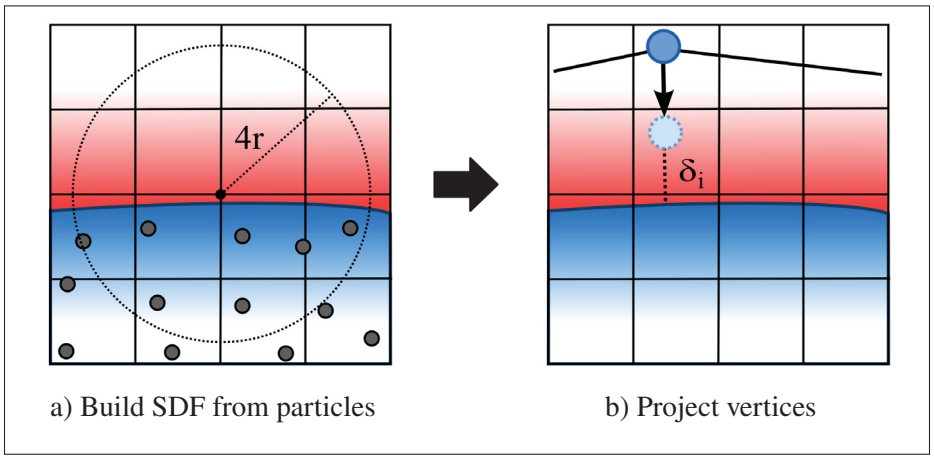


Figure 2.6 Detail-preserving projection steps. First, a signed distance field is built from the particles; then, it is used to project the vertices, while preserving the initial distance δ_i to the surface

on the isosurface $\phi = 0$, except in the special case where it lies exactly on a stationary point of ϕ. Furthermore, the distance between that point and the mesh vertex is equal to $\phi(\mathbf{x}_i)$, with a negative distance indicating that the vertex lies inside the implicit surface. These properties will be used during projection. To construct ϕ, the signed distance values around the isosurface $\phi = 0$ are first computed from the particles. Then, the distance information of interface cells is propagated further away by using the fast marching method of Sethian (1995).

The mesh vertices are relatively close to the implicit surface, and computing the signed distance field is rather costly. So, we only compute it locally, based on the previous maximal distance

between the mesh and the implicit surface. We thus use 1.25 times the previous maximal distance in our examples.

While several methods for surfacing particle fluids could be used to compute the values around the isosurface $\phi = 0$, the method described by Solenthaler *et al.* (2007) is used in our approach. In that method, an implicit function value is computed at each grid vertex position $\mathbf{x}_i$ using the average position of the neighbor particles inside an influence radius. In order to prevent erroneous results between distant particles, a correction is also applied based on the maximum eigenvalue of the average position gradient. In our explicit mesh surface examples, an influence radius of $4r$ is used to provide a smooth surface. The values computed using the method of Solenthaler et al. correspond to a signed implicit function, but do not form a signed distance field. Thus, they are converted by first triangulating the isosurface, and then setting the interface cells' value $\phi(\mathbf{x}_i)$ using their distance to the nearest triangle.

2.3.2 Projection

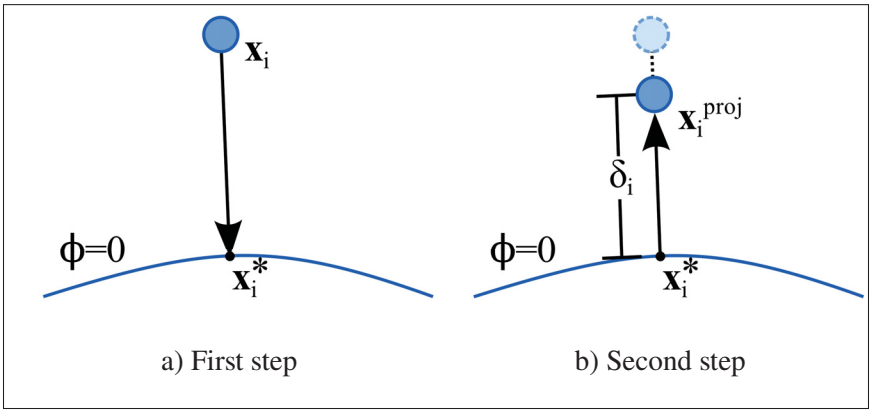


Figure 2.7 Our two-step projection: (a) First, the projection $\mathbf{x}_i^*$ on the isosurface $\phi = 0$ is computed; (b) then, the final projected position $\mathbf{x}_i^{proj}$ is found by moving the vertex along the projection line to match the distance δ_i

As stated previously, the implicit function, which is a signed distance field, and its gradient are used to project the explicit mesh vertices, in contrast to the method of Yu *et al.* (2012), which used their normals. Furthermore, an initial distance δ_i between a vertex and the isosurface $\phi = 0$

should be maintained during the projection in order to preserve surface details. This distance is computed at time $t = 0$ for all initial mesh vertices, and at creation time for vertices created later during the animation. By definition, in a signed distance field, $-\nabla\phi(\mathbf{x}_i) \cdot sign(\phi(\mathbf{x}_i))$ points in the direction of the closest point on the isosurface $\phi = 0$. Also, the value $|\phi(\mathbf{x}_i)|$ is equal to the distance from $\mathbf{x}_i$ to that point. Thus, the projection $\mathbf{x}_i^{\text{proj}}$ of $\mathbf{x}_i$ on the isosurface $\phi = \delta_i$ could be obtained using the following equation:

$$\mathbf{x}_i^{proj} = \mathbf{x}_i - \frac{\nabla\phi(\mathbf{x}_i)}{\|\nabla\phi(\mathbf{x}_i)\|} \cdot (\phi(\mathbf{x}_i) - \delta_i)\,. \tag{2.1}$$

However, our experiments showed that Eq. 2.1 often fails to find a good solution when $\mathbf{x}_i$ lies inside a cell containing a local maximum or minimum of ϕ because the grid interpolation approximation systematically under- or over-estimates the value of ϕ in these cells. To improve the precision in such cases, the proposed approach uses a two-step projection, where the closest point on $\phi = 0$ is first found (see Fig. 2.7a), and then the vertex is moved along the projection line to match the distance δ_i (see Fig. 2.7b). As the value of ϕ can be unreliable, the projection on $\phi = 0$ is found by iteratively moving toward $\phi = 0$:

$$\mathbf{x}_i^* = \mathbf{x}_i^* - \epsilon \cdot \frac{\nabla\phi(\mathbf{x}_i)}{\|\nabla\phi(\mathbf{x}_i)\|} \cdot \phi(\mathbf{x}_i),$$

where $\mathbf{x}_i^*$ is initially set to $\mathbf{x}_i$. The coefficient ϵ represents the fraction of the distance to travel toward $\phi = 0$, so it should have a value in the range (0,1], and is set to 0.35 in our explicit mesh surface examples. This operation is repeated until $\phi(\mathbf{x}_i^*)$ is lower than a predefined threshold ($0.005\Delta x_\phi$ is used in our examples, where Δx_ϕ is the cell size of ϕ). During this stage, a cubic interpolation is used to compute the value of $\phi(\mathbf{x}_i)$ in order to provide a better approximation of the surface curvature during the projection. Afterward, the final projection position $\mathbf{x}_i^{\text{proj}}$ is computed using the following equation:

$$\mathbf{x}_i^{proj} = \mathbf{x}_i^* + \frac{\mathbf{x}_i - \mathbf{x}_i^*}{\|\mathbf{x}_i - \mathbf{x}_i^*\|} \cdot \delta_i.$$

After these two steps, the vertex should be correctly projected on the surface, while preserving its initial distance.

2.3.3 Topological Changes Detection for the Implicit Function

In order to correctly remesh the explicit surface (Sec. 2.5), two main types of topological changes need to be accounted for: those in the implicit surface and those in the explicit surface. Topological changes of the implicit surface are handled in this section, while those of the explicit surface will be dealt with in Sec. 2.5. The changes in the topology of the isosurface $\phi = 0$ that we want to detect here are those that cause the surface to merge or split. Such changes result in a sudden "jump" in the vertex position when the projection is applied, allowing the detection of these topology changes with the following threshold:

$$\left|\mathbf{x}_i - \mathbf{x}_i^{proj}\right| > \tau.$$

When a vertex exceeding this threshold is found, it is moved back to its original position $\mathbf{x}_i$, and the cells (of the Γ grid) containing the neighbor triangles of that vertex are flagged for the topology matching (see Sec. 2.4) and the remeshing (see Sec. 2.5) operations. In our explicit mesh surface examples, $\tau = 0.5r$.

This approach is different from the method used in the work of Yu *et al.* (2012). In their case, vertices were detected when the projection failed to find the isosurface $\phi = 0$. Our projection is guaranteed to find a point on the isosurface $\phi = 0$, and as a result, it is necessary to use the threshold-based detection outlined above.

2.4 Optimization-Based Topology Matching

The goal of the topology matching stage is to handle problems outlined in Fig. 2.8, where the surface should merge (see Fig. 2.8a) or should disconnect (see Fig. 2.8c). These problems are caused by the divergence between the explicit and the implicit surface topologies, as can be seen in Fig. 2.9. Thus, it is crucial to intelligently adjust the explicit mesh to reflect the topological

changes from the implicit function, while preserving its details. This will be done with an optimization that locally adjusts the topology, and globally preserves details. The topology matching stage consists of two steps. First, using the technique described by Wojtan *et al.* (2010), a signed distance field is built from the explicit mesh, and stored in the mesh implicit representation grid Γ. Then, an optimization is performed on the grid to locally adjust the topology of Γ to mirror the topological changes of ϕ, which have previously been identified during the projection (see Sec. 2.3.3). Note that the optimization is performed only if one or more cells have been flagged during projection. This modified mesh implicit reprensentation grid will later be used to rebuild the mesh locally (see Sec. 2.5).

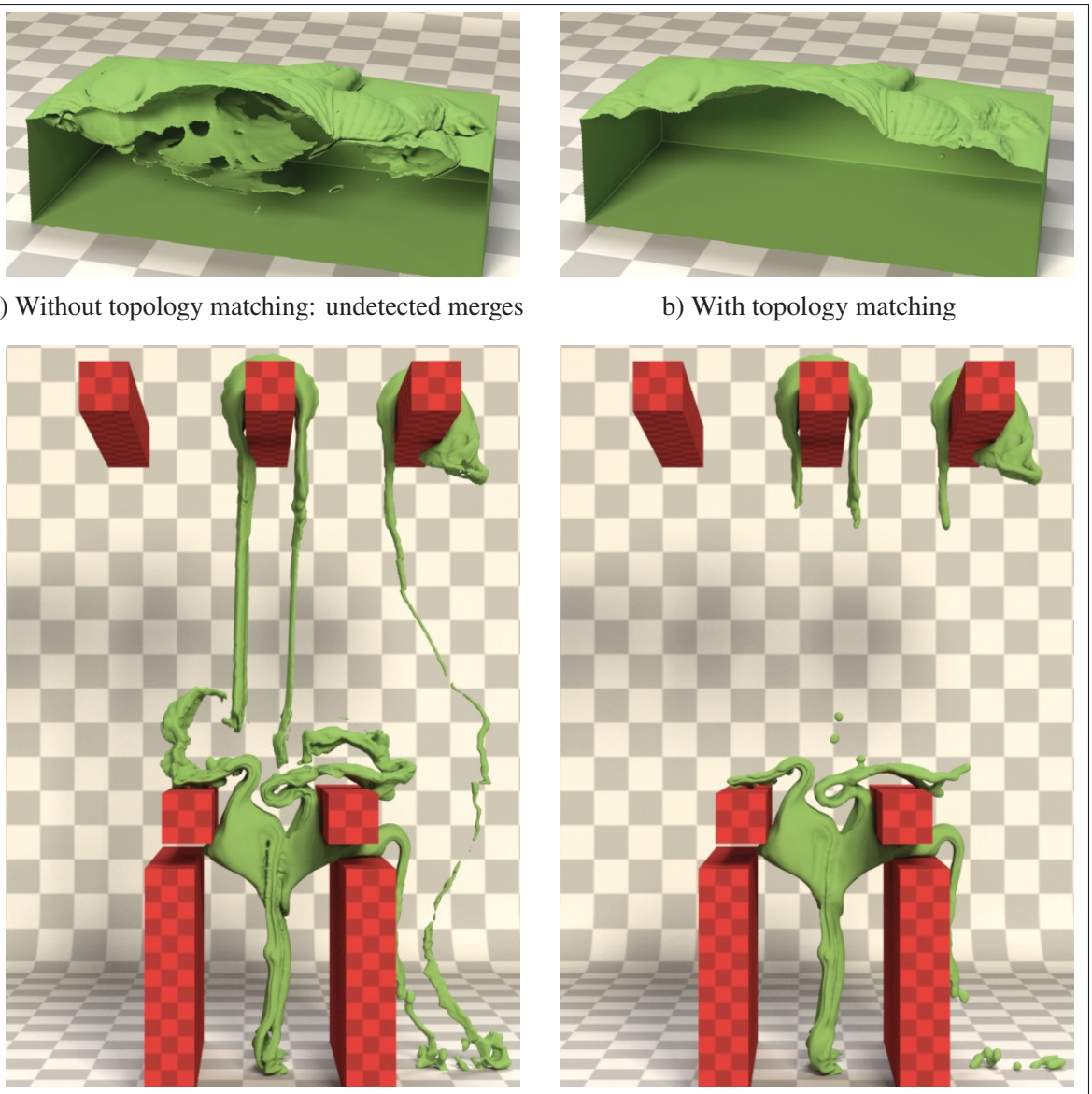
During the optimization, cells that are near the interface and have not been flagged during projection are marked as *fixed cells*. This flag indicates that these cells should not be modified by the topology matching stage. To be considered near the interface, the cell's distance from the surface should not be greater than the diagonal length of a grid cell:

$$|\Gamma_i| \leq h\sqrt{3},$$

where h is the grid cell size. Afterward, all the other cells are modified to match the topology of the implicit surface, and to provide a smooth transition with the *fixed cells*.

The main idea behind the topology matching operation is that, in *non-fixed cells*, the interface $\Gamma = 0$ should lie as much as possible on the same isosurface of ϕ. Thus, the difference between the value of Γ and ϕ in those cells should be similar to that of their neighbors. This will ensure that the values for those cells of Γ follow the implicit surface topology while being consistent with the *fixed cells*. Let us define the scalar field ψ as the difference between the explicit surface implicit representation Γ and the implicit function ϕ:

$$\psi(\mathbf{x}) = \Gamma(\mathbf{x}) - \phi(\mathbf{x}). \tag{2.2}$$



a) Without topology matching: undetected merges

b) With topology matching

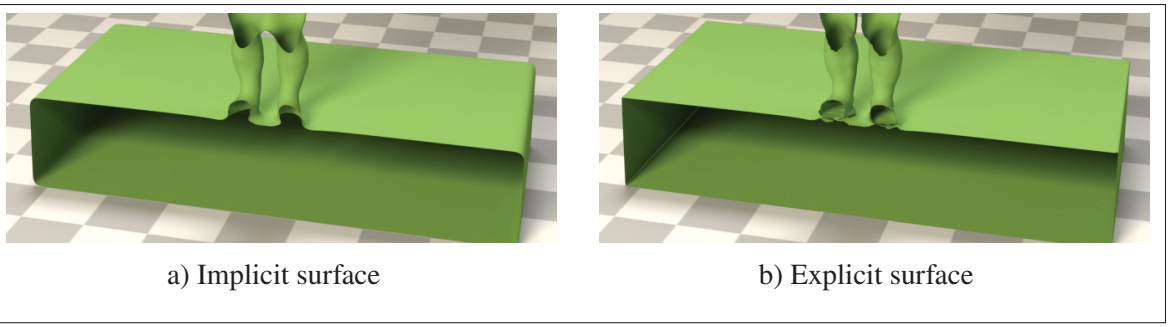
c) Without topology matching: undetected splitting

d) With topology matching

Figure 2.8 Topology matching is an important step of our approach, as can be seen in the results without (left) and with (right) this stage.

The objective is to minimize the variation of $\psi(\mathbf{x})$ around the neighborhood of $\mathbf{x}$. This can be expressed as the minimization of the Dirichlet energy function:

$$\min_{\psi} \frac{1}{2} \int \|\nabla\psi\|^2 dx,$$



a) Implicit surface b) Explicit surface

Figure 2.9 Cutaway view showcasing the divergence between the implicit and the explicit surface topologies, when topology matching is not performed

which is equivalent to solving the following Laplace equation:

$$\nabla \cdot \nabla \psi = 0. \tag{2.3}$$

The topology matching operation works by solving Eq. 2.3 for every *non-fixed cell*, as will be explained in the following sections.

2.4.1 Topology Matching Solver

Solving Eq. 2.3 first requires discretizing the scalar field ψ in a grid with the same dimensions, position, and size as Γ. Thus, each vertex of ψ corresponds to a vertex of Γ, with the same position and indices. From that, Eq. 2.3 is reformulated using the central difference:

$$6\psi_{i,j,k} - (\psi_{i-1,j,k} + \psi_{i+1,j,k} + \psi_{i,j-1,k} + \psi_{i,j+1,k} + \psi_{i,j,k-1} + \psi_{i,j,k+1}) = 0. \tag{2.4}$$

This equation is satisfied when $\psi_{i,j,k}$ is equal to the average value of its neighbors, resulting in a smooth scalar field. Thus, the value of ψ will be smoothly interpolated between *fixed cells*.

Putting the previous equation into matrix form gives:

$$\begin{pmatrix} 6 & \alpha_{21} & \cdots & \alpha_{1n} \\ \alpha_{21} & 6 & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & 6 \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where

$$\alpha_{ij} = \begin{cases} -1 & \text{if } j \text{ is a neighbor of } i. \\ 0 & \text{otherwise.} \end{cases}$$

This is a simple linear system in the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a sparse symmetric positive definite matrix. Solving that particular type of equation is a well-known problem, and it can be carried out efficiently using a wide range of solvers. We validated our approach with both a *modified incomplete Cholesky conjugate gradient level 0 (MICCG(0))* solver and a *successive over-relaxation (SOR)* solver. While the *MICCG(0)* solver performed better with a larger number of cells, in most of our examples the speed of the *SOR* solver was similar or better because of its lower time cost per iteration. Furthermore, this solver does not require any matrix construction, which therefore lowers its memory requirement. Thus, a *SOR* solver was used in our explicit mesh surface examples. This solver works by iteratively updating the value of each grid vertex ψ_i using the following equation:

$$\psi_i^{l+1} = (1-\omega)\psi_j^l + \frac{\omega}{\alpha_{ii}} \left(-\sum_{j<i} \alpha_{ij}\psi_j^{l+1} - \sum_{j>i} \alpha_{ij}\psi_j^l \right). \tag{2.5}$$

The relaxation factor ω is used to accelerate convergence, and a value in the range of $0 < \omega < 2$ guarantees convergence. Choosing its value carefully can increase the performance of the solver. Fortunately, Yang & Gobbert (2009) showed that the optimal value ω_{opt} for the 3D Poisson equation, which is a generalization of Laplace's equation, with Dirichlet boundary conditions can be computed using the following equation:

$$\omega_{opt}(N) = \frac{2}{1 + \sin\left(\frac{\pi}{N+1}\right)},$$

where N is the dimension of the grid, assuming a $N \times N \times N$ grid. Since our grid does not necessarily have uniform dimensions, we use the maximum dimension along one axis. This equation was used to compute the relaxation parameter value in all our explicit mesh surface examples.

The solver is iterated until the maximum relative difference gets below a threshold:

$$\max_i \|\psi_i^{l+1} - \psi_i^l\| < \beta.$$

The threshold value $\beta = 0.01h$ is used in all the examples.

Algorithm 2.1 shows the complete topology matching stage. During the first step of this stage, ψ_i is initialized for each cell using $\psi_i = 0$. Moreover, the *fixed cells* of Γ are identified and the Dirichlet boundary conditions are enforced for the cells in ψ (see Sec. 2.4.2). Note that since the resolution of grid ϕ is not necessarily the same as Γ, linear interpolation is used to compute the value of ϕ at each grid vertex. Afterward, the successive over-relaxation iterations are computed until convergence is reached. Finally, Γ is updated using the new values of ψ.

Note that the successive over-relaxation solver does not require a temporary structure to store the previous values of ψ^k. This is because ψ_i^{k+1} is computed using the previously updated value when the neighbor has already been processed.

2.4.2 Boundary Conditions

For cells lying on the grid boundaries, we enforce Neumann boundary conditions by copying their values inside the missing neighbors:

$$\frac{\partial \psi_i}{\partial x} = \frac{\partial \psi_i}{\partial y} = \frac{\partial \psi_i}{\partial z} = 0$$

Algorithm 2.1 Topology matching

```
1  Procedure TOPOLOGY_MATCHING
2      forall cells i in Γ do
3          if (i is not flagged) AND (||Γ_i|| ≤ (h√3)) then
4              Set i as a fixed cell
5              ψ_i = Γ_i − interpolate(x_i, φ)
6          else
7              ψ_i = 0
8          end if
9      end forall
10     while (max_i ||ψ_i^{l+1} − ψ_i^l|| > β) do
11         forall non-fixed cells i do
12             Compute ψ_i^{l+1} using Eq. 2.5
13         end forall
14     end while
15     forall non-fixed cells i do
16         Γ_i = ψ_i + interpolate(x_i, φ)
17     end forall
```

For example, if a cell $\psi_{i,j,k}$ has no neighbor on its left, we set $\psi_{i-1,j,k} = \psi_{i,j,k}$. The left hand side of Eq. 2.4 then becomes:

$$\begin{aligned} 6\psi_{i,j,k} - (\psi_{i-1,j,k} + \psi_{i+1,j,k} + \psi_{i,j-1,k} + \psi_{i,j+1,k} + \psi_{i,j,k-1} + \psi_{i,j,k+1}) &= \\ 6\psi_{i,j,k} - (\psi_{i,j,k} + \psi_{i+1,j,k} + \psi_{i,j-1,k} + \psi_{i,j+1,k} + \psi_{i,j,k-1} + \psi_{i,j,k+1}) &= \\ 5\psi_{i,j,k} - (\psi_{i+1,j,k} + \psi_{i,j-1,k} + \psi_{i,j+1,k} + \psi_{i,j,k-1} + \psi_{i,j,k+1}) &\quad . \end{aligned}$$

It can be seen that each missing neighbor decreases the coefficient of α_{ii} by exactly one. Thus, the condition can be enforced by modifying the way it is computed:

$$\alpha_{ii} = \mathcal{N}_i,$$

where $\mathcal{N}_i$ is the number of neighbor cells of i.

In order to preserve the original surface inside the *fixed cells*, Dirichlet boundary conditions are enforced. The values $\psi_{i,j,k}$ of those *fixed cells* are initialized using Eq. 2.2, and are not modified by the solver.

2.5 Explicit Surface Topology and Meshing

The topology of the explicit mesh implicit representation (Γ) and the topology of the implicit function (ϕ) now match, and the cells of Γ requiring remeshing based on changes in the implicit surface topology are marked for reconstruction. However, the explicit mesh surface itself might contains self-intersections or thin stretched geometry that would require remeshing in features that are too small to be detected by the implicit representation. We use the method of Wojtan *et al.* (2009) to mark such cells. We also use their method for the reconstruction of the mesh geometry inside all the marked cells. We will briefly explain their method here. The topological changes of the explicit mesh surface are first detected by marking cells where multiple interpenetrations occur in the explicit mesh. A "deep cell test" is then used to compare the topology of the explicit mesh and its implicit representation Γ, in order to detect regions where merging and splitting occurred. Afterward, in cells marked during both the topology matching stage and the identification of topological changes of the explicit mesh surface, the geometry is removed from the explicit mesh, and then reconstructed using the marching cube algorithm from Lorensen & Cline (1987). In our implementation, the modified look-up table of Montani, Scateni & Scopigno (1994) is used to resolve ambiguous cases and avoid producing cracks in the resulting geometry. Note that the detection and the surface reconstruction are based on the adjusted value of the explicit mesh implicit representation Γ, ensuring a smooth transition with the surrounding detailed explicit mesh.

2.6 Implementation

As is typical in explicit mesh methods (Brochu & Bridson, 2009; Wojtan *et al.*, 2009, 2010; Yu *et al.*, 2012), after the mesh is advected, edges that are too small are collapsed and those that are too long are split. Also, as in the method of Yu *et al.* (2012), edges are collapsed when the dihedral

angle of the supporting triangles is too large. These operations help maintain a good mesh quality throughout the surface tracking process. As in previous methods (Brochu & Bridson, 2009; Wojtan *et al.*, 2010), the modified butterfly subdivision scheme of Zorin, Schröder & Sweldens (1996) is used to position the new mesh vertices during splitting operations. During a collapse operation, if a vertex lies on a ridge or a corner, its position is used, otherwise, the butterfly subdivision is used instead, such as in Brochu & Bridson (2009). We refer the reader to the course notes of Wojtan *et al.* (2011) on mesh-based surface tracking for a more detailed explanation of how to preserve a good mesh quality.

The stitching mechanism of Wojtan *et al.* (2010) is used when connecting the newly created mesh with the old one. This method better preserves the details in those regions than that of Wojtan *et al.* (2009), and is less prone to the generation of a non-manifold geometry. Furthermore, as in the work of Wojtan *et al.* (2009), when a non-manifold geometry is detected, the neighbor cells are marked and reconstructed.

2.7 Results

We tested our approach in a range of scenarios, ranging from a slowly melting object to severe deformations induced by rigid obstacles that tear apart objects made of viscous material. Several test scenarios can be found in the accompanying video of our paper (Dagenais *et al.*, 2017), and representative frames are shown in Fig. 2.10. We first show that surface details are well preserved over time for low deformation rates by slowly melting the Stanford bunny from its base (see Fig. 2.11), and dropping a highly viscous dragon on obstacles (see Fig. 2.2, Fig. 2.10b, and the paper's video. The bunny and the dragon were simulated using only 27,952 particles and 20,456 particles, respectively, and yet our approach preserved the original details of the mesh surface throughout the simulation, while the implicit surface did not.

We then show that the approach correctly handles merging fluids by dropping viscous objects on top of one another. The cutout views in Fig. 2.12 and in the our paper's accompanying video show that the topological changes are correctly handled as the objects merge together, ensuring

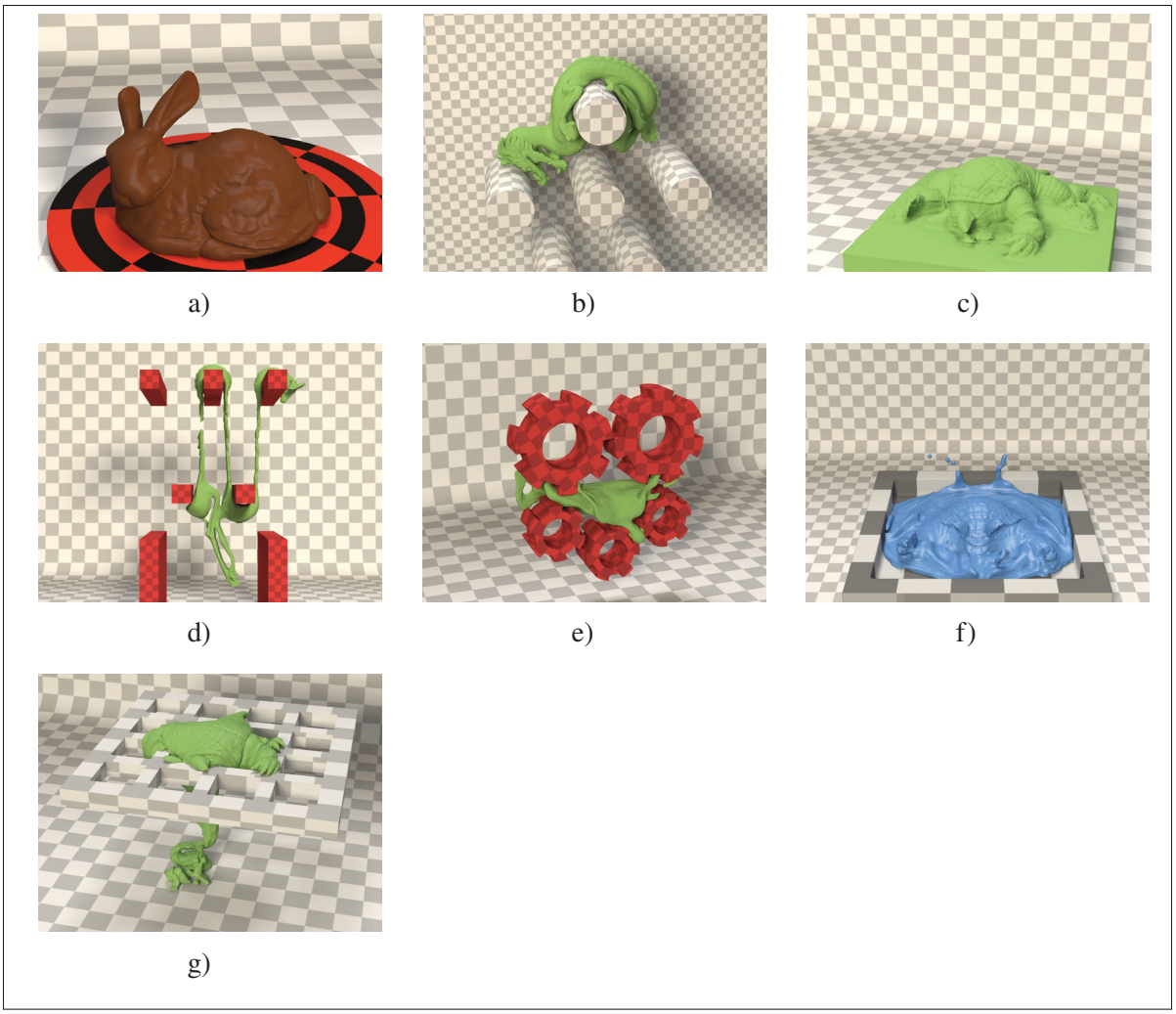


Figure 2.10 We tested our approach on several scenarios, ranging from slow deformation to a large number of complex topological changes

that the topology of the explicit mesh surface is consistent with that of the implicit surface. With our approach, the finer details of the original mesh surface, such as the horns of the gazelle and the armadillo's shell, are well preserved (see paper's accompanying video). Similarly, we show that our approach handles fluid splitting correctly by tearing apart objects (see Fig. 2.13, Fig. 2.10e, and the paper's accompanying video). Surface splitting is handled gracefully by our approach, consistent with the implicit surface topology. Moreover, surface details are well preserved in regions that have not been affected by topological changes, such as the head of the gazelle.

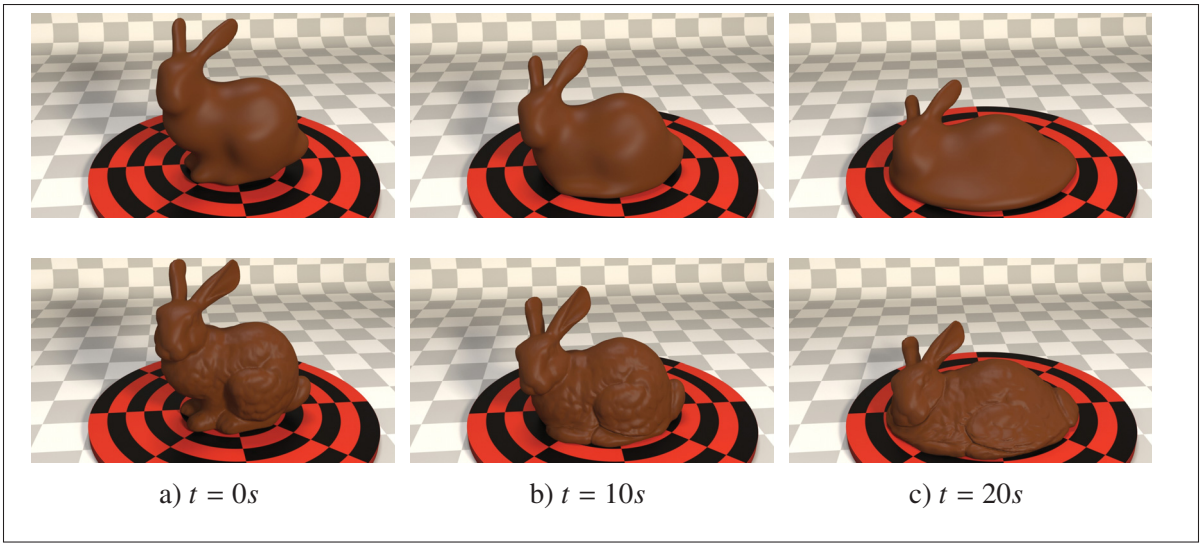


Figure 2.11 Melting of the Stanford bunny from the bottom using only 27,952 particles. Top: the implicit surface built from the particles; bottom: our explicit surface

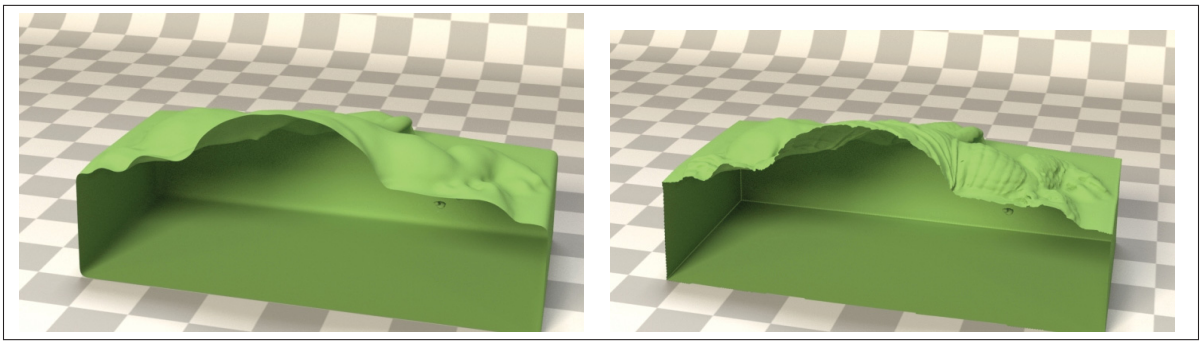


Figure 2.12 Viscous objects falling on top of one another at time $t = 4s$ (269,257 particles). Left: cutout view of the implicit surface built from the particles; right: cutout view of our explicit surface

We show that the approach handles cases where multiple and complex topological changes occur by simulating a non-viscous liquid (see Fig. 2.14). Surface details, such as the face, the teeth, and the shell of the armadillo, are preserved until reconstruction occurs in those areas. Furthermore, the surface is stable when multiple splittings and mergings occur at once. Even the splashes are captured correctly, thanks to our topology matching stage.

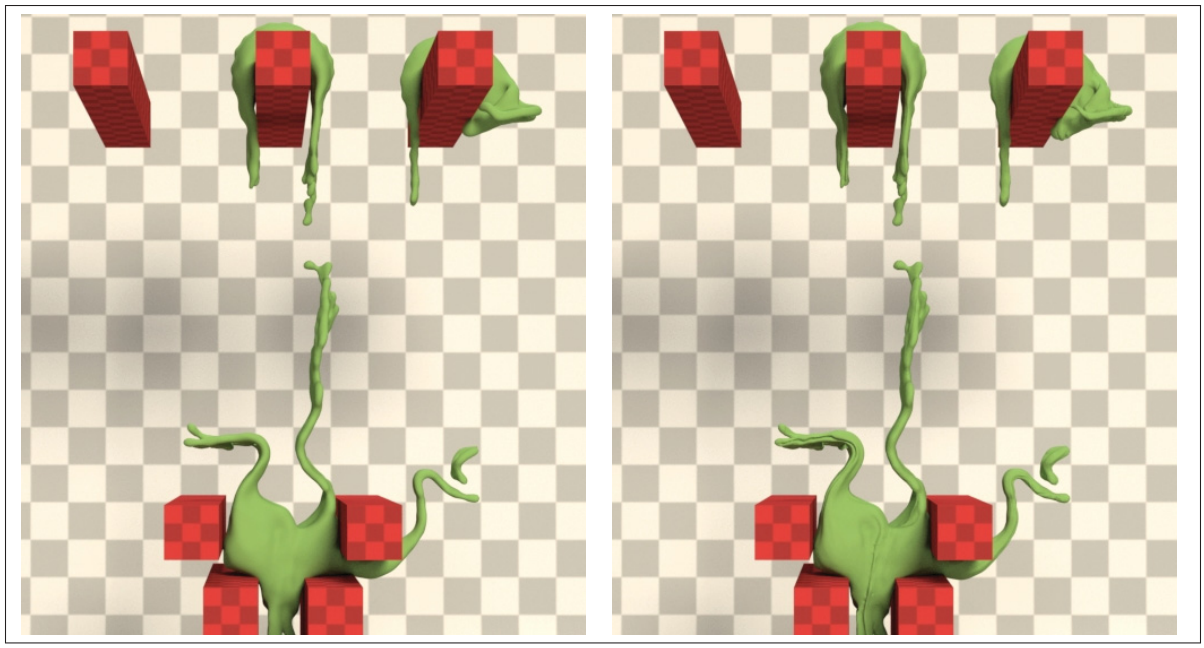


Figure 2.13 Tearing apart an object at time $t = 2s$ (21,362 particles). Left: the implicit surface built from the particles; right: our explicit surface

In Fig. 2.15, two liquid bunnies collided before falling into a pool of liquid. Despite the complexity of the topological changes, the resulting explicit mesh surface closely follows the implicit surface. Note that the complete sequence is available in the paper's accompanying video.

We compared the results using a coarser and finer resolution for the explicit mesh implicit representation Γ in Fig. 2.16 and in the paper's accompanying video. When the resolution is too coarse, finer details, such as the horns of the gazelle, might be erroneously split. Furthermore, the reconstruction might not be accurate enough, which can result in loss of details, and in lumps floating around the object. On the other hand, as can be seen in Table 2.1, the computation times of the topology matching and the explicit topology & meshing stages increase considerably when a finer resolution is used. The examples shown in this chapter use $\Delta x_\Gamma = 0.5r$, except for the ones shown in Fig. 2.16. Smaller details could be better preserved, even with a coarser grid, by using the approach of Wojtan *et al.* (2010) to handle the explicit surface topological changes. However, since that technique does not reconstruct the surface directly from the

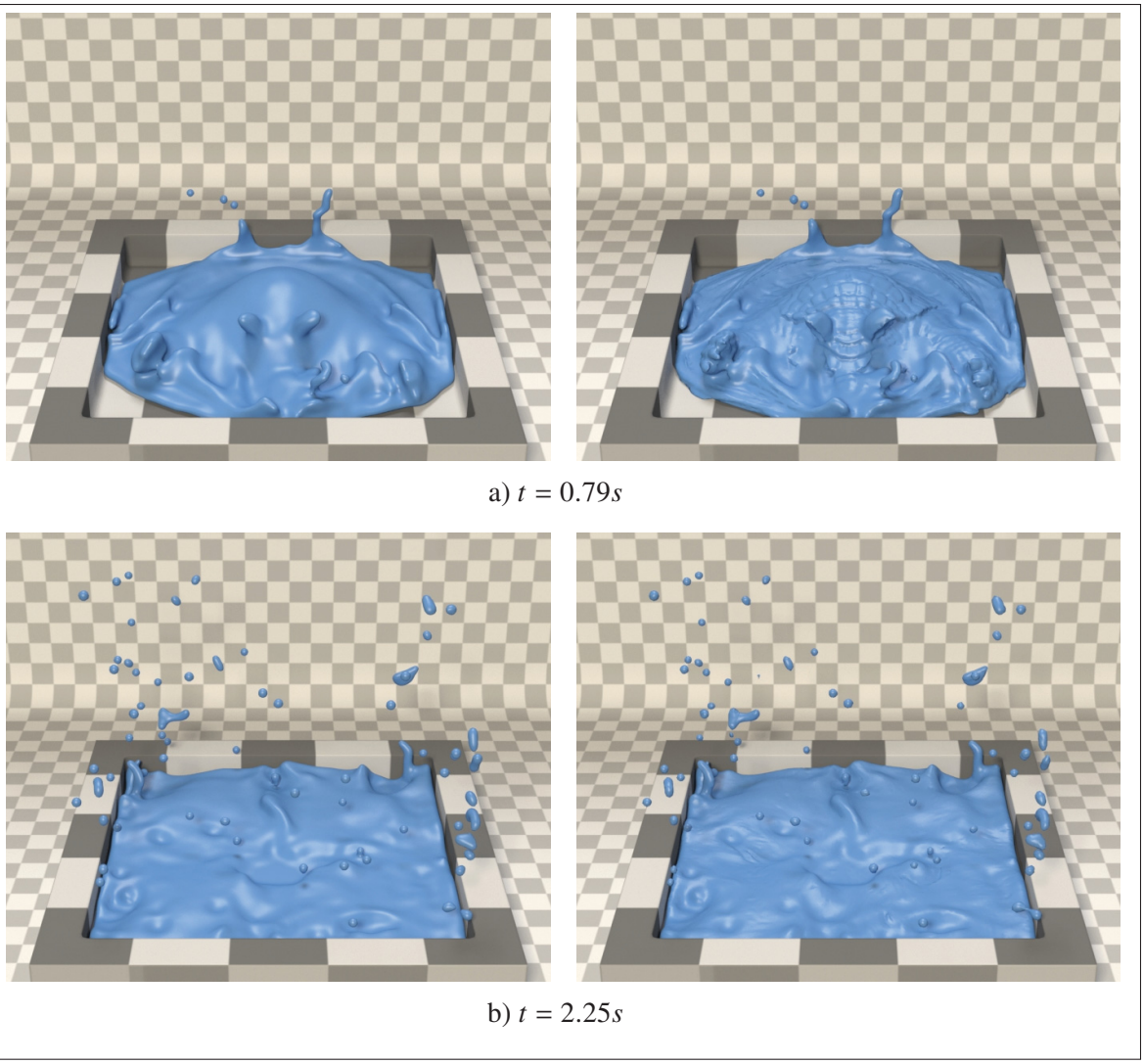


Figure 2.14 Non-viscous liquid Stanford armadillo (35,214 particles). Left: the implicit surface built from the particles. Right: our explicit surface

implicit representation, it would also require changing the way our topology matching stage works. Similarly, we validated our scenarios with different resolutions for the implicit function ϕ. We found that using a coarser resolution, such as $\Delta x_\phi = r$, resulted in small oscillations of the implicit surface that where transferred to the explicit surface during animation. Using a finer resolution $\Delta x_\phi = r/2$ removed the oscillations.

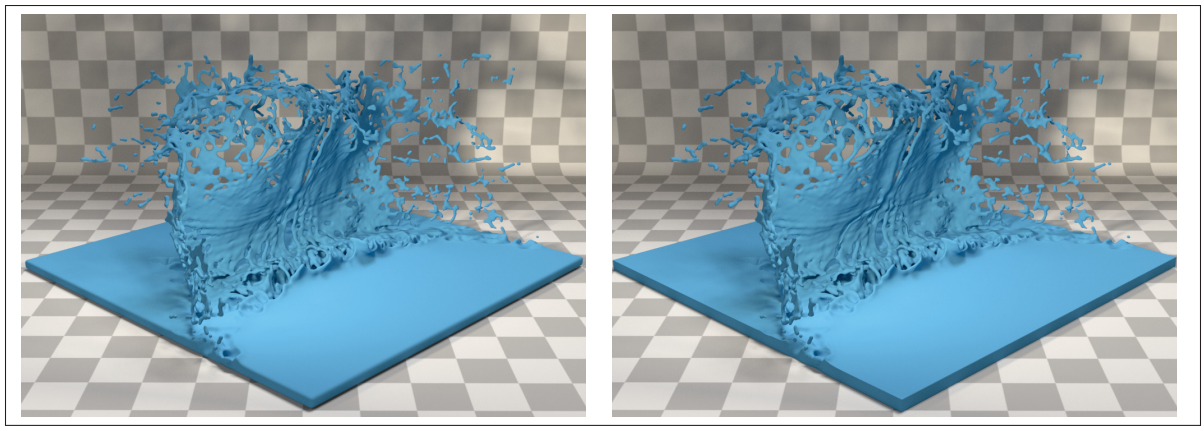


Figure 2.15 Splashes generated after two liquid bunnies collided. Top: the implicit surface; bottom: our explicit mesh surface

Table 2.1 Timing comparison for the gazelle scenario, with different resolutions Δx_Γ

	Topology matching	**Explicit topology & meshing**
$\Delta x_\Gamma = 0.25r$	155.79 s	36.91 s
$\Delta x_\Gamma = 0.5r$	8.28 s	5.11 s
$\Delta x_\Gamma = 1.0r$	0.53 s	1.90 s
$\Delta x_\Gamma = 2.0r$	0.05 s	1.11 s

The timings of all the examples are shown in Table 2.2, and the relative time for each part of the approach is depicted in Fig. 2.17. The most time-consuming parts of our approach are the computation of the implicit surface (Solenthaler *et al.*, 2007), the topology matching stage, and the explicit topology & meshing. Furthermore, the ratio of the computation times for these parts is not always the same. As can be seen in Table 2.2, the computation time of the implicit surface is proportional to the number of particles, but it should be noted that the number of particles does not affect the computation times of the other parts. The timings of the advection, projection, and explicit topology & meshing stages depend mostly on the number of triangles. Moreover, the computation times of the explicit topology & meshing stage are proportional to the amount of topological changes. The topology matching stage also requires more time to compute for scenarios where the surface is contained within a large volume (splitting gazelle, falling objects, and gears). As the number of cells increases, the matrix system becomes larger,

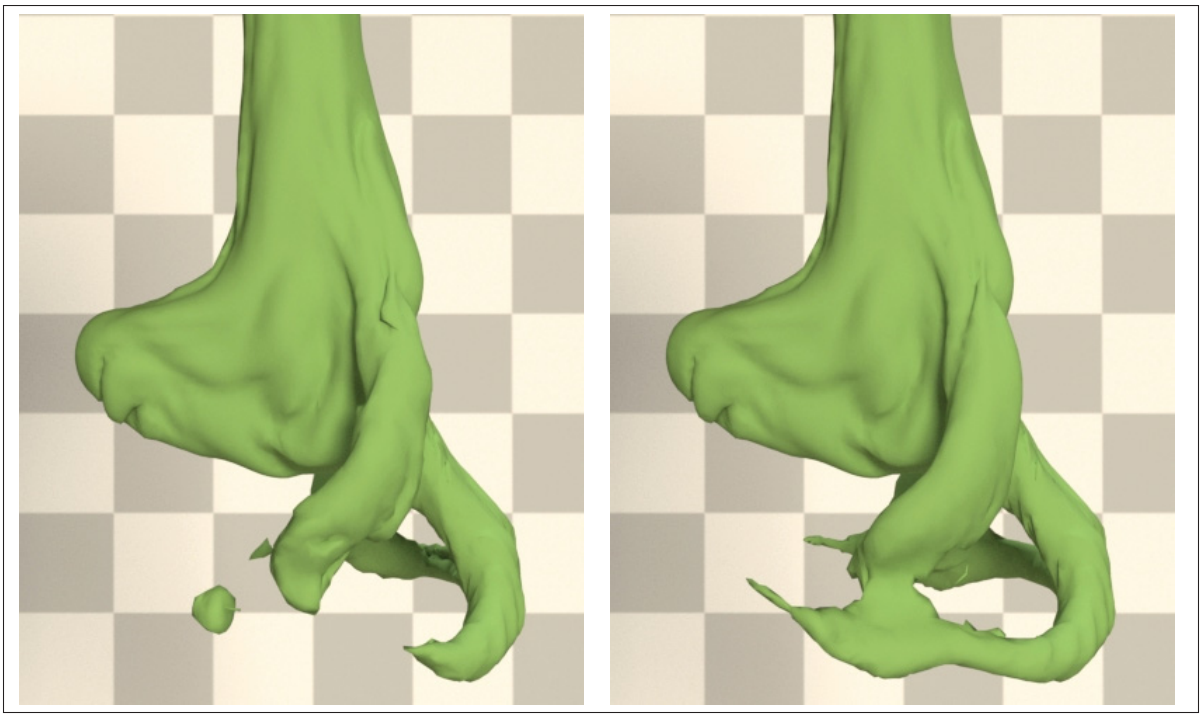


Figure 2.16 Comparing our results with the resolution $\Delta x_\Gamma = r$ (left) and $\Delta x_\Gamma = r/2$ (right)

and the solver requires more iterations to convergence. However, the impact this stage has on the overall time is greatly reduced as this operation is skipped when no early topological changes are detected (see Sec. 2.3.3). This can be observed in scenarios with few topological changes (bunny, falling objects, and viscous armadillo), where it was performed on less than 50% of the total frames. We believe computation times for this operation could be further reduced by limiting the processing to cells within a threshold distance to the *flagged cells*. However, care must be taken not to choose a distance that is too small, which might prevent distant volumes from merging, and would result in holes on the surface.

In our approach, all the computations are performed only once per frame. This is different from the case of Yu *et al.* (2012), where the advection and the topological changes were performed at every timestep of the simulation. In their case, this is necessary since an accurate surface representation is needed throughout the simulation in order to compute accurate surface tension forces. Other papers based on an Eulerian simulation (Wojtan *et al.*, 2009; Müller, 2009; Wojtan

Table 2.2 Timings for all explicit mesh surface tracking examples. The per frame average computation times are for the explicit mesh advection (Sec. 2.2), the implicit surface evaluation and triangulation using the method of Solenthaler *et al.* (2007) (Sec. 2.3.1), the signed distance field calculation (evaluating the implicit function, and then performing the FMM (Lorensen & Cline, 1987) (Sec. 2.3.1), the detail-preserving projection (Sec. 2.3.2 and Sec. 2.3.3), the topology matching stage (Sec. 2.4), and the explicit topology & meshing stage (Sec. 2.5). Numbers in parentheses correspond to the percentages of frames where the topology matching stage was executed

			Avg. time per frame (s)						
Example	**Avg. # particles**	**Avg. # triangles**	**Advect**	**Implicit function**		**Proj.**	**Topology matching**	**Explicit topology & meshing**	**Total**
				Eval	**SDF**				
Dragon (Fig. 2.2)	20,456	257,785	1.17	3.01	1.39	1.24	1.59 (58%)	3.16	11.55
Gears (Fig. 2.10e)	8,877	121,414	0.39	1.82	3.06	0.48	13.48 (83%)	5.55	24.77
Viscous armadillo (Fig. 2.10g)	35,195	85,274	0.41	5.34	3.43	0.32	4.13 (43%)	3.59	17.22
Melting bunny (Fig. 2.11)	27,952	66,862	0.30	3.98	0.92	0.24	0.00 (0%)	1.48	6.92
Falling objects (Fig. 2.12)	259,752	370,789	1.29	35.96	9.60	1.10	8.18 (38%)	7.27	63.40
Splitting gazelle (Fig. 2.13)	21,362	242,932	0.87	3.43	2.53	0.81	8.28 (83%)	5.11	21.03
Liquid armadillo (Fig. 2.14)	35,214	100,870	0.41	6.04	4.15	0.67	5.43 (94%)	4.51	21.20

et al., 2010) mention that they can handle topological changes only once per frame; however, they still need to advect the surface at every timestep since they need it to update the interior cells of the simulation. Even though the advection stage is not costly, this dependency makes it harder to recompute the surface without re-running the simulation. Since we use a particle-based simulation, no information from the surface is needed during the simulation. Therefore, with our approach, it is possible to generate the explicit mesh surface as a post-process operation once the entire simulation has been computed. This enables a more user-friendly workflow, in which the user can obtain the desired behavior from the simulation, and then separately generate and adjust the explicit mesh surface without affecting the simulation.

All the simulations shown in this chapter were generated using Houdini's FLIP fluid solver, except for the liquid Stanford armadillo example (see Fig. 2.14), where an SPH simulator was

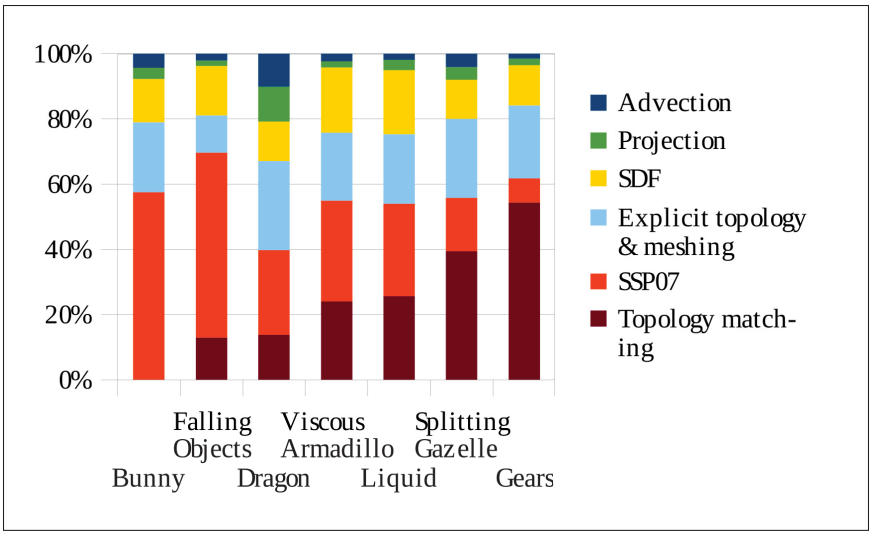


Figure 2.17 Timing percentages for all explicit mesh surface tracking examples. The per frame timing percentages are for the same stages as those given in Table 2.2

used (Becker & Teschner, 2007). Particles resampling was disabled in the FLIP solver to prevent particles from being added or removed near the surface, which results in minor flickering of the implicit surface. Our approach could take advantage of parallel computing; however, the timings shown in this chapter are for a sequential execution on a single core. The timings information provided in this section are for a 3.2 GHz Intel i7-3930K CPU with 32 GB of RAM. The images were rendered using Houdini's Mantra.

2.7.1 Comparison

We implemented the method of Yu *et al.* (2012), and compared it to our approach. We show the comparison with the dragon example, which contains a high amount of small geometrical features on the surface, as well as larger features that extend further away from the the implicit surface, such as the scales on the back and tail. As can be seen in Fig. 2.18, their method fails to preserve surface details of the original mesh, because of their projection on the implicit surface. Our approach successfully preserves such surface details. In our experimentations, we also observed that the projection of Yu *et al.* (2012) is unstable, and introduces flickering of the surface in regions with high curvature if it is calculated only once per frame (see paper's

accompanying video). In contrast, our projection proved to be stable using only one step per frame, which allows us to handle topological changes only once per frame.

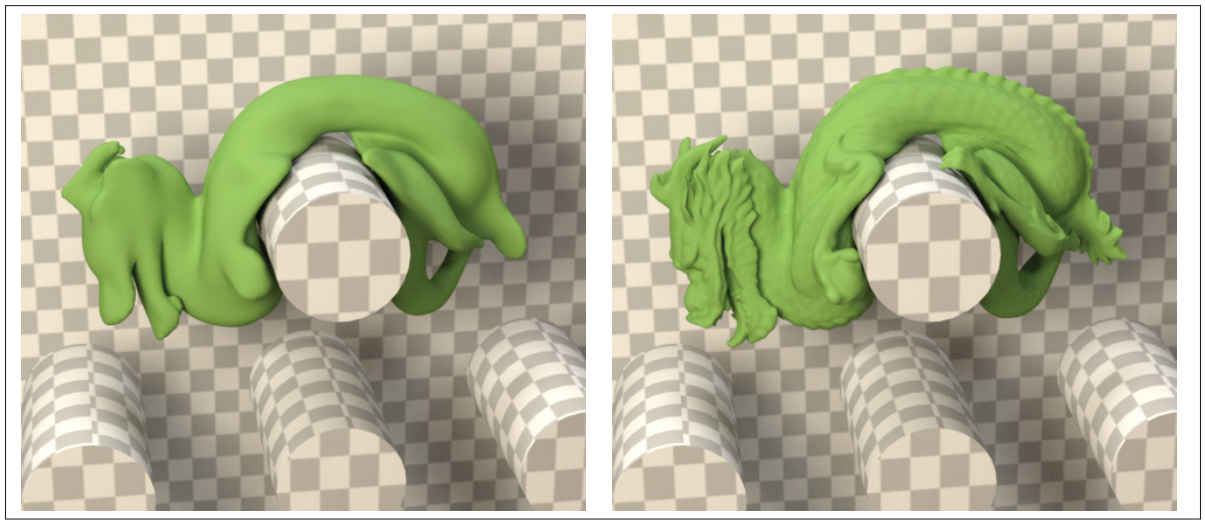


Figure 2.18 Comparison between the technique of Yu *et al.* (2012) (left) and our approach (right). Our projection preserves the details of the initial explicit mesh surface

2.8 Discussion

In this chapter we presented a new detail-preserving projection, coupled with a novel topology matching stage. Unlike previous methods, our approach preserves the original surface details, even when the underlying simulation is very coarse. The approach was successfully used with simulations of melting objects, highly viscous objects, and even non-viscous liquids. It was used with a FLIP and an SPH simulator, and could easily be adapted to any particle-based simulator.

One downside of using an explicit mesh surface is the temporal discontinuities that may occur when large regions of the explicit mesh surface are reconstructed. These discontinuities may be in the form of a surface *popping* artifact on some frames of the animation, and may be caused by a coarse grid resolution (see Fig. 2.16 and the paper's accompanying video), or by a late detection of small topological changes on the implicit surface. Nonetheless, in the first case, a smaller implicit representation grid cell size reduces this problem, at the cost of longer computation times. As for the second case, the problem occurs when a small topological

change on the implicit surface does not induce a vertex movement of the nearby mesh that is large enough to be detected by the threshold described in Sec. 2.3.3. The reconstruction is delayed until the threshold is triggered, which forces the reconstruction of a larger region of the explicit mesh. This problem generally occurs when small holes are created in a thin volume of liquid. This can be seen in the viscous armadillo and in the colliding bunnies scenarios, where holes in the implicit surface are not immediately detected and handled, resulting in a delayed reconstruction of a larger region of the mesh. This can be reduced by using a smaller threshold τ when detecting topological changes in the implicit function. However, choosing a value that is too small might unnecessarily reconstruct portions of the mesh where no topological changes of the implicit function occurred. Furthermore, changes in topology that occur further from the explicit surface, such as the creation of bubbles and droplets, may not be detected by our criterion. However, these features are transferred to the mesh implicit representation Γ during the topology matching stage. Because this stage is triggered only when a topological change is detected by the criterion, in some cases it may require a few frames before the change in topology is reflected on the explicit mesh. A compromise is to perform the topology matching stage at every frame, at the cost of longer computation times. The problem of identifying topological changes of an implicit surface is not new, and more robust approaches could be used instead of our criterion. For instance, it can be shown from Morse theory that changes in the topology of a parameterized implicit surface occur at the location of a critical point (Hart, 1998). By tracking such events using interval arithmetic, Stander & Hart (1997) were able to identify topological changes of an implicit surface. While such an approach would add a significant overhead to our computation times, we believe it could allow the early detection of holes, bubbles, and splashes. One could also use concepts from digital topology to detect such changes. Using a level-set evolution such as that of Han, Xu & Prince (2003), which preserves topology, our implicit surface could be morphed into the explicit mesh implicit representation while preserving its topology. The conflicting regions between the two surfaces could then be used to identify where their topologies differ. It should be noted that such a technique would increase computation times, and that developing a robust solution might not be as trivial as expected. The evolution of

the implicit surface needs to be carefully directed, as multiple solutions can be topologically equivalent at the global scale, but might not reflect the same local topological changes.

While our approach preserves well the finer surface details, this might not be entirely desirable in some cases such as with non-viscous liquids. This can be seen in the liquid armadillo (see Fig. 2.14) and splashing bunnies (see Fig. 2.15) examples, where small wrinkles are preserved on the liquid surface as it comes to rest, and small folds are sometimes preserved where two surfaces merge. A smoother surface is expected for such a non-viscous liquid. Decreasing the distance δ_i between the vertices and the isosurface should result in the expected behavior, as the explicit surface would then take on the appearance of the isosurface. A metric based on the viscosity and the deformation undergone by the surface, e.g., the magnitude of the pressure gradient, such as in the work of Goldade, Batty & Wojtan (2016), could be used to determine where and when to adjust δ_i. Additionally, while our explicit surface can handle the frequent remeshing caused by the multiple topological changes that non-viscous liquids generally undergo, it tends to lose details at points where topological changes of the implicit surface occurred. This is caused by the topology matching stage, which tries to replicate the shape of the implicit surface in regions that are being reconstructed. Nonetheless, our approach is still relevant for non-viscous liquids, as the explicit mesh surface preserves its original shape until it undergoes several topological changes, and can also be used to track surface properties such as color or physical quantities, such as in the work of Yu *et al.* (2012).

By using an explicit mesh surface representation, it is also possible to use textures to add much finer visual details on the surface. These details can take the form of colors, normals, and even displacements. Using our approach, texture coordinates are well preserved until topological changes occur. However, once the surface is reconstructed, it is not trivial to find valid texture coordinates that will not introduce excessive stretching or compression of the texture. To avoid this issue, it could be possible to generate a new coherent texture in regions that undergo topological changes using a texture synthesis approach such as dynamic lapped textures (Gagnon, Dagenais & Paquette, 2016; Gagnon *et al.*, 2019; Gagnon, Guzmán, Mould & Paquette, 2021).

In the future, it would be interesting to extend the approach to preserve thin sheets even when using a coarse grid resolution. We would also like to extend the approach to better handle non-viscous liquids in order to preserve the surface details and smooth them out as the surface changes. Furthermore, this approach could be extended to accurately track surface properties such as color or texture coordinates. Lastly, it would be interesting to combine our approach with methods that can control the animation by matching user-provided mesh surfaces at specific times during liquid simulations (Sivakumaran, Paquette & Mould, 2022).

CHAPTER 3

FLUID SIMULATION FOR SNOW IMPRINTS

The work described in this chapter has been published in the *The Visual Computer* journal (Dagenais, Gagnon & Paquette, 2016):

François Dagenais, Jonathan Gagnon, and Eric Paquette. *An Efficient Layered Simulation Workflow For Snow Imprints*, The Visual Computer, Volume 32, Number 6, pp 881–890, 2016.

As part of this publication, my work was focused on the base layer and particle simulation, as well as the application of gravity on the snow mist. My colleague's work focused on the sourcing and simulation of the snow mist.

In the previous chapter, details were handled mostly at the mesh surface level, allowing a coarser simulation to be run without compromising too much the amount of features on the mesh surface. Other techniques for improving computation times of a simulation without compromising too much its quality in term of visual features make use of multi-resolution simulations, or even mix different simulation types. In this chapter, we apply the latter concept to settled snow, which remains mostly static until it interacts, directly or indirectly, with a collision object.

Snow is an important aspect that is ubiquitous in movies with winter scenery. While modeling static snow can be an easy task, simulating its interaction with characters and dynamic objects requires much more time and efforts in order to achieve a visually realistic behavior. Simulators available in current commercial software and state of the art simulation methods for snow and granular material (Ihmsen *et al.*, 2013; Macklin *et al.*, 2014; Narain *et al.*, 2010; Stomakhin *et al.*, 2013; Zhu & Yang, 2010) can simulate snow at a limited scale. They require a large amount of simulation data to generate realistic results even just at a moderate scale, which results in a large memory consumption and long computation times. Furthermore, in most scenes, but especially in large scenes, a considerable amount of the memory and computation will most likely involve snow that remains unchanged throughout the simulation. Typical scenarios, like character footsteps or sleigh tracks, interact with a small subset of snow. In such cases where

the interactions with the snow are localized, height map methods (Onoue & Nishita, 2005; Pla-Castells *et al.*, 2008; Zeng *et al.*, 2007) limit their computations to areas affected by the motion of dynamic objects. Relying on a much more efficient simulation strategy, these methods significantly reduce the memory and computation costs. However, they are limited to a simple deformation of a flat surface, thus they cannot model snow that is projected in the air.

This chapter introduces an efficient workflow to handle the interaction between snow and dynamic objects by subdividing the phenomenon into three separate simulations (see Fig. 3.1). First, our base layer simulation uses constructive solid geometry (CSG) operations on level set representations of the snow and the dynamic objects to leave smooth imprints and determine locations where finer resolution simulations are locally required. Then, a modified granular simulation adds snow particles to model the behavior of snow that gets pushed and tossed. Finally, a tailored fluid simulation replicates the look and the behavior of the airborne sparse snow mist driven by the surrounding air medium. The contributions of our approach are:

- A decoupling of the behavior of the snow into three components;
- An efficient simulation approach for each of the three components;
- Adapted simulations taking into account the impact of the air resistance for the snow particles and mist;
- A control over the transfer of snow from the base layer to the snow particles and then to the snow mist.

Given our cascade of simulation components, our approach provides early feedback of the look of the imprints and the movement of the snow particles. It handles the snow interaction with simple or complex dynamic objects, it can cope with large scenes, and it provides good visual results. Furthermore, it significantly decreases the memory consumption and the computation times by adding details only where they are required. Finally, it provides separate control over three different components of the snow behavior.

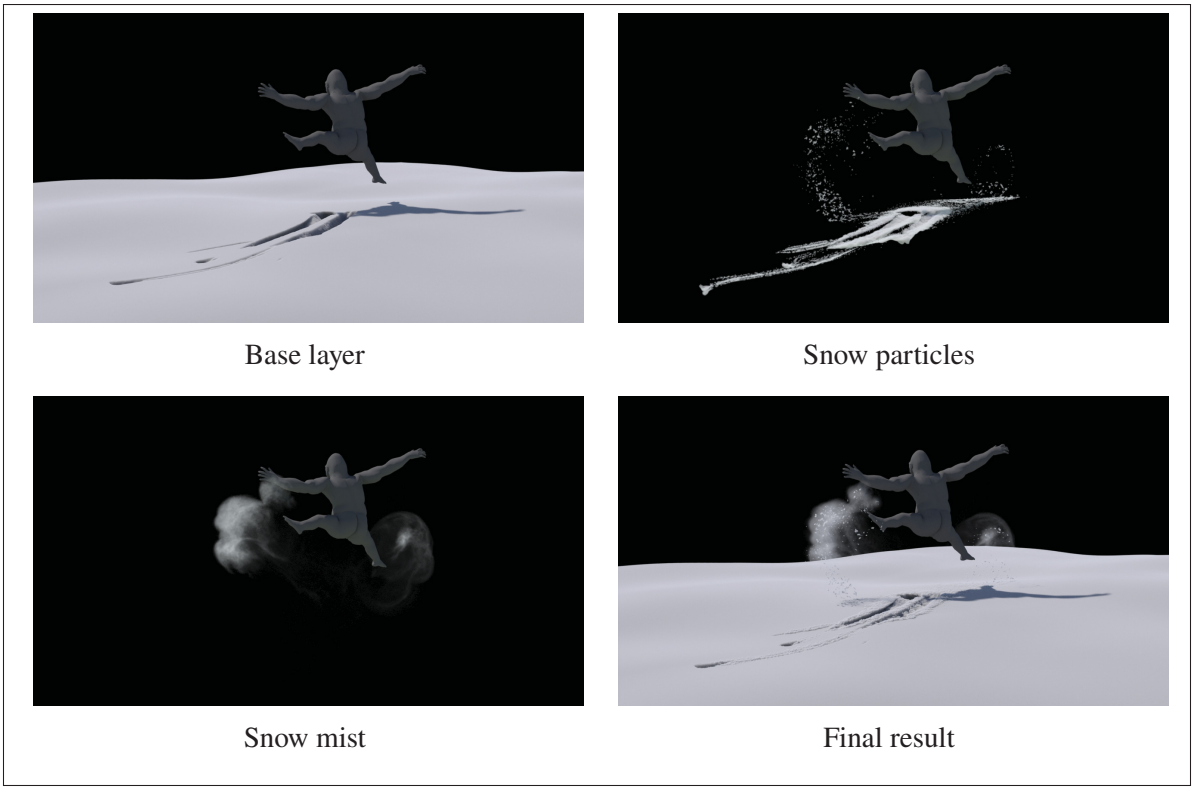


Figure 3.1 Our decomposition into three components allows us to model efficiently the fine scale details of snow

3.1 Overview of the Proposed Snow Imprints Workflow

Our approach aims to simulate powdery snow, which gets compressed under the weight of objects, such as a character's feet, and also exhibits complex behaviors when it interacts with objects or becomes airborne. As shown in Fig. 3.2, to achieve such a behavior we decompose the physical process into three components: the base layer (see Sec. 3.2), the snow particles (see Sec. 3.3), and airborne snow mist (see Sec. 3.4). We use separate simulation paradigms for each of the components; When brought together, these three components provide a very efficient and easy to control snow imprint workflow.

The first component, which we call the base layer, is made of the snow that did not interact with the dynamic objects yet. Handling it is efficient, as it is processed only when the objects collide with it. In such event, the snow of the base layer is compressed by the dynamic objects, and some

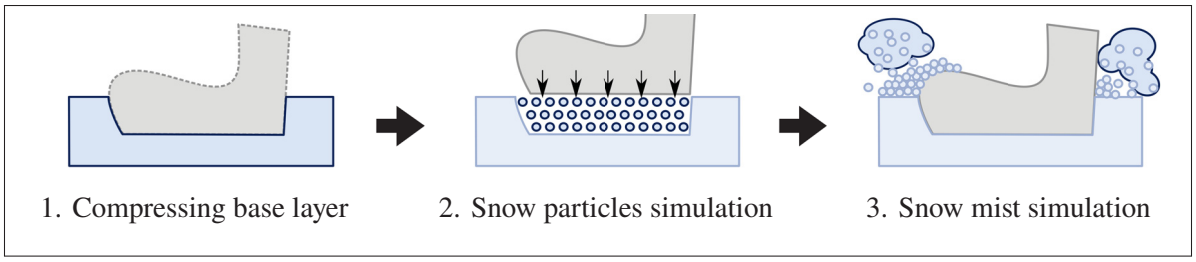


Figure 3.2 Our approach consists of three combined simulations, each related to a different component of our physically-based process decomposition

of that snow is transferred to the second component: the snow particles. Those moving snow particles reproduce the complex behavior of snow that interacts with the dynamic objects, other snow particles, and the base layer. Driven by a particle simulation, they enable the accumulation of snow as well as the projection of snow in the air. While these snow particles mimic the behavior of snow clumps of modest size, they fail to recreate the motion of tiny airborne snow elements, whose behavior is strongly affected by the supporting air medium. Hence, the airborne snow particles trigger the creation of the third component: the snow mist. This last component relies on an incompressible fluid simulation, and its behavior relies on that of the surrounding air medium.

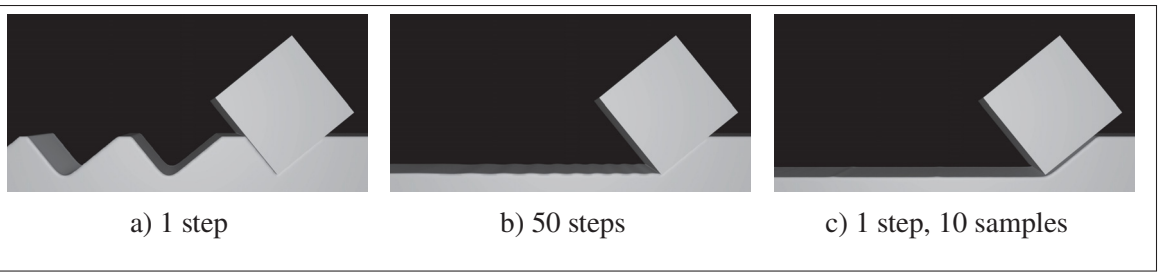
To maintain a realistic and efficient simulation, we rely on a unidirectional coupling between the three components; the base layer influences the snow particles, which in turn influence the snow mist. This brings the advantage of providing early feedback: the base layer simulation can be entirely computed before simulating the snow particles, which can in turn be completely fine tuned before generating the snow mist.

3.2 Base Layer

The initial snow layer can be designed in two ways: by specifying a height on top of the objects and ground, or by specifying the top surface of the snow. The base layer is constructed from this designed snow shape. Then, throughout the animation, the parts that collide with the dynamic objects are removed to match the contour of the volume swept by the objects. Part of this

volume loss will be compensated for by the added particles in the dynamic particles simulation (see Sect. 3.3.2), while the remaining reduction replicates the compression of the snow under external forces. To have a fast computation, we found that applying CSG operations on a level set (Museth, Breen, Whitaker & Barr, 2002) provides the versatility needed to model realistic imprints of the dynamic objects. This level set is updated, once per frame, by compacting the regions swept by the dynamic objects. In order to do so, a second level set is built using the dynamic objects, and is then subtracted from the base layer. If no special treatment is done, "dents" will be visible because of the discrete positions in time. This problem can be seen in Fig. 3.3a, where a box is quickly moved through the base layer of snow. Performing multiple substeps can reduce this problem. However, a large number of substeps is required for animations with fast moving objects (see Fig. 3.3b), which greatly increases computational time. To overcome this limitation, the dynamic objects' geometry is sampled at times $t + n\frac{\Delta t}{N}$, where $n = 0, 1, 2, ..., N$, N is the number of samples, t is the time at the previous frame, and Δt is the time difference between two animation frames. These samples are individually converted to a level set, and then combined using a union operation (see Fig. 3.4b). In order to fill the gap between the samples, a smoothing operation is performed on the level set. Because this operation generally shrinks the level set, a dilation is performed before the smoothing operation to compensate for the volume loss. Besides, this dilation operation also helps to better connect the samples. The result of these operations is a smooth level set spanning the whole animation path (see Fig. 3.4c). This outlines an advantage of our approach compared to methods relying on height fields: both the base layer and the volume swept by the dynamic objects share a unified representation that makes it easy to compute operations such as CSG union and subtraction as well as smoothing and dilation. It should be noted that the number of samples is much lower than the number of simulation substeps needed to obtain similar results (see Fig. 3.3b and 3.3c), thus making this approach much faster.

In reality, snow imprints rarely have the exact shape of their "source" object. Some snow is slightly pushed during the impact, and some is taken off as the object moves away. Moreover, snow does not react as a completely homogeneous material and it often contains small air pockets.



a) 1 step b) 50 steps c) 1 step, 10 samples

Figure 3.3 A moving box sweeping through the base layer. (a) With only one step per frame, the full motion of the object is not well captured. (b) Even with a large number of simulation steps, small "dents" can be seen. (c) Our approach provides smoother results with only one step and a few samples

Leaving a sharp and precise imprint does not result in a realistic imprint. To improve the realism of our approach, the dynamic objects' level set is updated with controllable irregularities before it is subtracted from the base layer (see Fig. 3.4d). To do so, we use a technique inspired by the level set surface modeling operators of Museth *et al.* (2002). These operators deform a level set by advecting it with a velocity field. This field is aligned with the gradient field of the level set, which forces the surface to move along its normal. In our approach, the length of the displacement is determined using a noise function:

$$\mathbf{v}(\mathbf{x}) = \frac{\nabla\phi(\mathbf{x})}{\|\nabla\phi(\mathbf{x})\|} \cdot \max(\text{noise}(\mathbf{x}), 0),$$

where $\phi(\mathbf{x})$ denotes the value of the level set evaluated at position $\mathbf{x}$, and $\nabla\phi(\mathbf{x})$ its gradient. To make sure the base layer will not enter the object, negative displacements are prevented. The level set of the objects is displaced using this velocity field, resulting in the final surface (Fig. 3.4d). Such noise has been added to all of the snow imprints examples, and its parameters have been determined experimentally to result in a reasonable size, considering the expected size of the snow clumps. Fig. 3.5 shows the improved realism of the base layer with such noise.

3.3 Snow Particles

At this point, the base layer models the basic shape of the imprints in the snow. However, it provides a very limited interaction with the dynamic objects. To better mimic the behavior

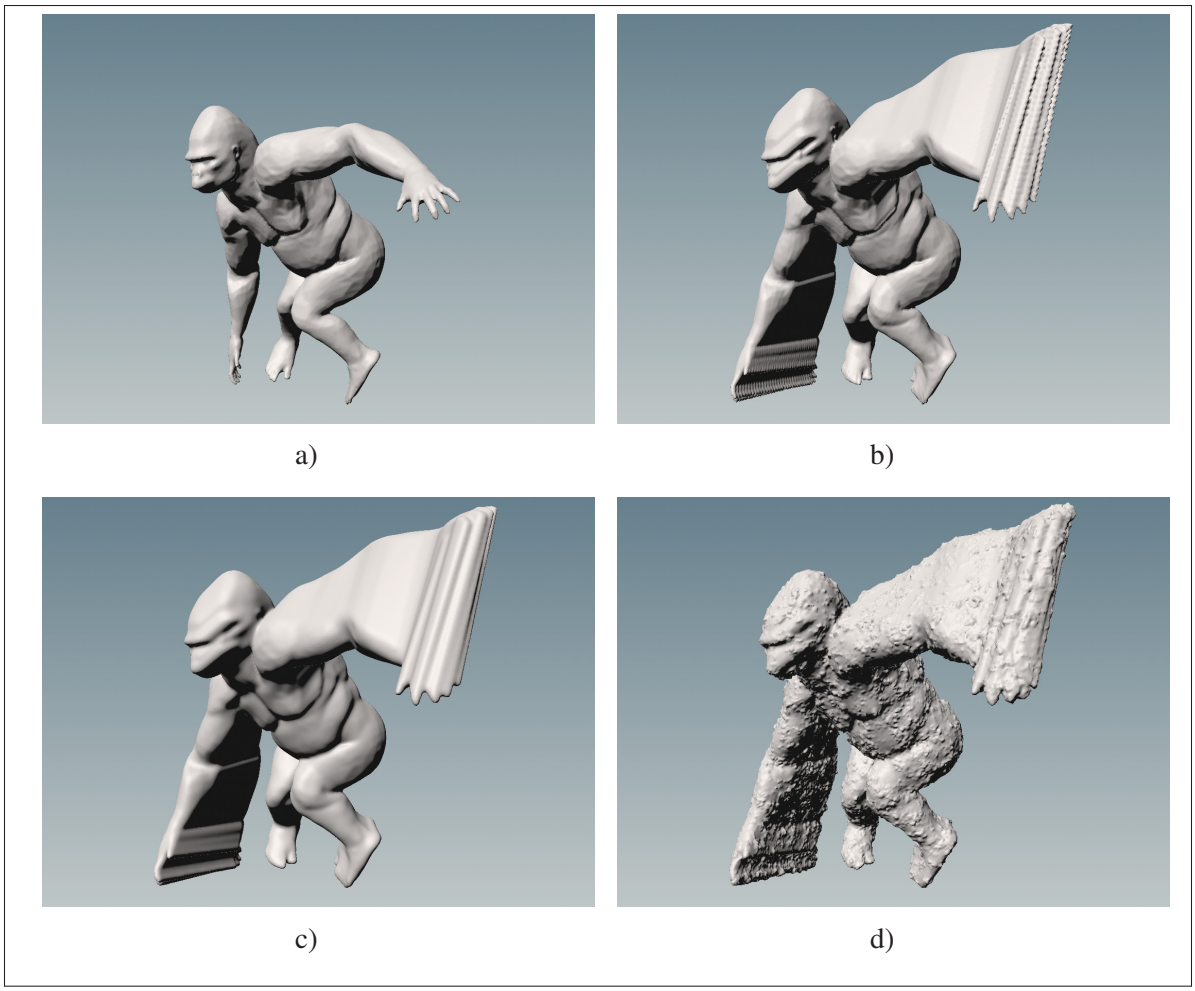


Figure 3.4 Before being subtracted from the base layer, dynamic objects undergo several operations. The original geometry (a) is sampled multiple times throughout its motion (b). It is then converted to a level set, before being dilated and smoothed (c). Finally, a noise displacement is applied on its surface (d)

of snow moved by these interactions, snow particles are used. While these particles interact with the base layer and the dynamic objects, they also interact with each other. Thus, they can be pushed, piled up, and even projected in the air, allowing for a more complex behavior that complements the imprints from the base layer.

a) Without noise

b) With noise

Figure 3.5 Comparison of the base layer rendered without noise (a) and with noise (b)

3.3.1 Granular Simulation

We opted for a granular material simulation that relies on position based dynamics (Macklin *et al.*, 2014; Müller *et al.*, 2007) as it can efficiently handle a lot of particles. Interactions between the snow particles and the base layer are handled by considering the base layer as a

collision geometry in the simulation. Furthermore, to the influence that gravity, collisions, attractions, and friction have on particles, we added the effect of air resistance:

$$\mathbf{v}_i^* = \beta^{\Delta t}\mathbf{v}_i,$$

where $\mathbf{v}_i$ is the particle's velocity, $\mathbf{v}_i^*$ is the modified velocity, and β is the air resistance coefficient in the range [0..1].

3.3.2 Particles Sourcing

As stated earlier, snow is not fully compressed when it collides with dynamic objects. This is considered in our approach by filling the volume of snow removed from the base layer with an amount of snow particles that is inversely proportional to the compression of the snow (see Fig. 3.6). This volume is computed by subtracting the base layer level set at time $t + \Delta t$ from that of time t using a CSG subtraction operation. The compression of the snow is controlled using a single parameter, the compression ratio, that varies between zero (incompressible) and one (fully compressible). Given the number of particles that can fit inside the removed volume, this number is diminished as the compression ratio is increased. We consider that snow is initially at rest, therefore the particles are initialized with a null velocity. The whole particle sourcing operation is done only once per frame before handling the snow particles dynamics. Thus, snow particles are created at time t, and then from time t to $t + \Delta t$ the dynamic objects are animated while the particles are simulated. As the granular simulation is computed, the dynamic objects collide with the particles, letting the collision response determine the behavior and the velocity of these new particles.

3.3.3 Inactive Particles

In our approach, computation times and memory consumption are considerably lowered by using a signed distance field, the base layer, to depict snow that remained still. This effectively lowers the number of particles to process. Still, their number can greatly increase as more particles are

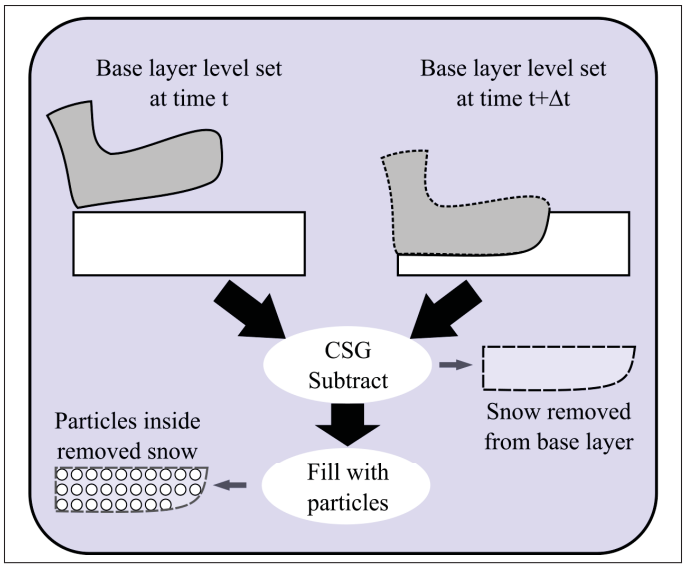


Figure 3.6 Snow particles sourcing. Snow compressed from the base layer is computed using a CSG subtract operation, and is then filled with snow particles

added from the interaction between snow and dynamic objects. This can significantly affect the computation times. Because most particles quickly come to rest and stay still, a lot of computation time is wasted. To avoid such unnecessary calculations, only particles near dynamic objects or near moving particles are computed. To determine which particles should be simulated, a level set is built from the particles whose velocity is higher than a predefined velocity threshold and from the dynamic objects. Then, the distances of each particle to this level set is computed. If this distance is further than a target distance threshold, the particle is considered inactive. These inactive particles are not processed during the particle dynamics computations. It should be noted that they are still rendered, and that they are used when computing collision, attraction, and friction forces of active particles. They can also be reactivated later when dynamic objects or moving particles get closer. In our examples, the list of inactive particles is updated twice per frame to reduce the likelihood of fast moving dynamic objects leaving the simulated particles region. Another option would be to use a larger target distance threshold, and to update the list only once per frame. However this would include more particles that may not collide with the objects.

Figure 3.7 In this photograph, snow mist is visible as tiny snow particles are projected in the air. This public domain image is provided by Adam Longnecker

3.4 Snow Mist

For completeness, this section describes the snow mist. However, this aspect of the approach was mainly the contribution of our co-author Jonathan Gagnon. Fig. 3.7 shows the three types of behavior modeled by our approach: the static base layer, moderate size snow clumps projected in the air (our snow particles), and the thin snow mist. Snow particles driven by the granular material simulation model pushed and projected snow clumps quite well, but it would require a tremendous number of tiny particles and long computation times to visually replicate the snow mist. Furthermore, snow mist requires a special treatment, as its behavior is highly affected by the surrounding air medium. The density distribution resulting from the turbulent motion typical of smoke simulations can be seen in the snow mist of Fig. 3.7. We therefore use another simulation paradigm to simulate snow mist in a more realistic way. Additionally, we have significant computation time gains by simulating this component separately from the others.

3.4.1 Snow Mist Simulation

We use an Eulerian incompressible fluid simulation for snow mist. Generally, in such simulation, gravity forces are not considered since the whole simulation domain is filled with gases of equivalent densities. However, since snow mist is much denser than the surrounding air, we apply gravity forces in the fluid simulation where the density is greater than 0. However, we use a lower value $g_{mist} = -2.0\ m/s^2$ in our examples to account for air friction.

3.4.2 Snow Mist Sourcing

Snow mist is sourced into the simulation from airborne snow particles, once per frame. To do so, potential snow particle candidates for sourcing are identified by testing their velocity against a threshold, to determine particles with high velocity. This criterion has been inferred from observations, and can be intuitively understood: higher velocity results in more energy and turbulence, which has a greater potential to break snow clumps into snow mist. In our examples the threshold value $\mathbf{v}_{\text{mist}} = 2.0\ m/s$ is used. Additionally, falling snow particles, i.e., $\mathbf{g} \cdot \mathbf{v} > 0$, are ignored to prevent snow mist creation when particles fall due to gravity. This way, snow mist is mostly generated from strong impacts with dynamic objects, which makes it look like finer snow particles are pushed in the air from the impact instead of being detached from snow chunks. This condition can be removed if the latter effect is desired. Using the candidate particles, a seed density field is filled from the union of spheres at each particles' center (radius is twice that of the particles). To increase the realism at a marginal computation cost, we also rely on the addition of a limited amount of noise to this seed density field. Similarly, a seed vector field is filled from the average velocity of the neighbor particles, where seed density has been added. Finally, both the seed density field and the seed velocity field are added in the simulation.

Our observations showed that mass transfer between snow particles and the snow mist is limited. Furthermore, as the snow mist constantly dissipates in the mist simulation, the unaccounted mass gain is counteracted by this dissipation. Thus, mass transfer between the snow particles and the snow mist is not taken into account in our approach.

3.5 Rendering

Before being rendered, the base layer is tessellated, making it easy to use with any surface shader. However, as it is quite common to render snow as a density volume (Stomakhin *et al.*, 2013), the base layer is rendered as a volume with constant density. Similarly, particles are also rendered as a density volume. This volume is built using the union of spheres located at each particle's center, with each sphere having the same radius r as the particles. A smooth density falloff of length $r/2$ is added around each sphere to make snow look softer, as the density decreases at the snow boundaries. Having both the base layer and the snow particles being rendered as volumes makes it easier to match visually the appearance of both layers, and also makes their boundaries seamless. As for snow mist, its density field is rendered as it is using a standard smoke shader.

3.6 Implementation

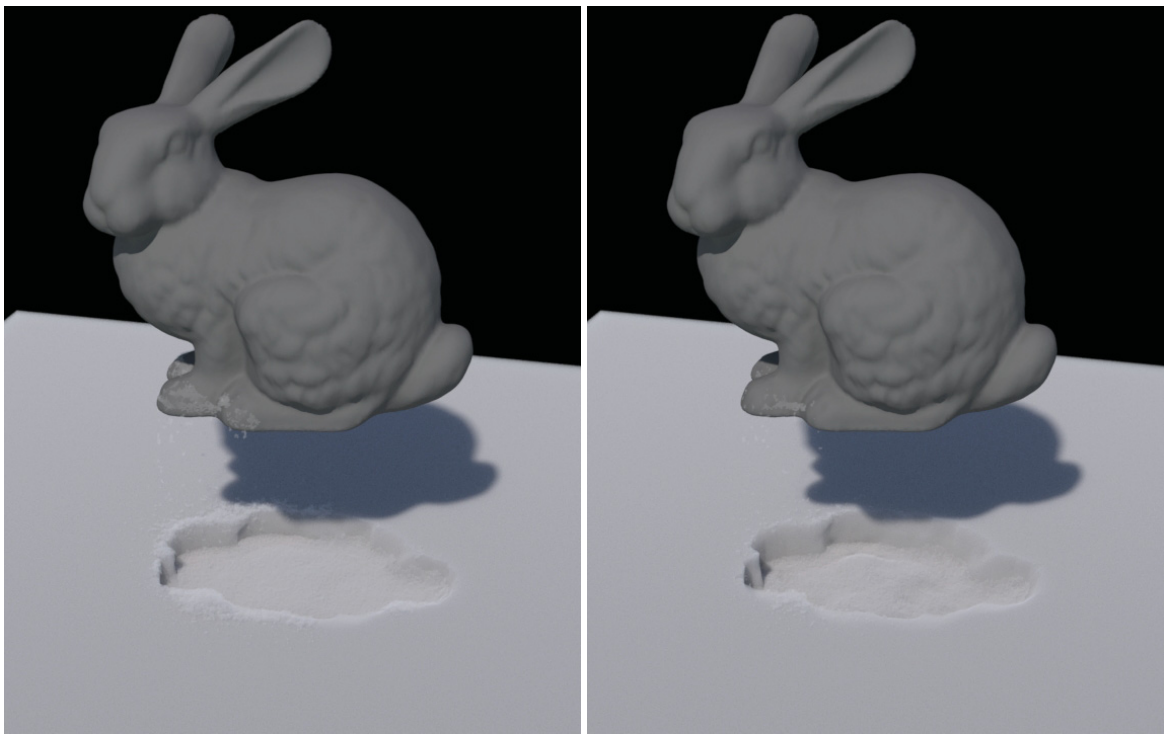
Our approach has been integrated inside Houdini™. We used OpenVDB (Museth *et al.*, 2013) to store level sets and apply operations on them, e.g., CSG, dilation, smoothing, and advection. The sparse representation used by this library helps to lower memory consumption and computation times considerably. The granular material simulator of Houdini™, which employs position based dynamics (Macklin *et al.*, 2014), was used to simulate snow particles. We extended their simulator to use our air friction model (see Sec. 3.3.1), instead of their force-based air friction, using Houdini's programming language called VEX. The snow mist solver is an adapted version of the smoke solver of Houdini™, to which our additional forces have been added. The solver uses a dynamic grid whose dimensions are adjusted based on the distribution of density in the scene. This greatly improved performances in our experiments. All our results were rendered using Houdini Mantra™. The *uniform volume shader* provided by Mantra™ was used for the base layer surface. Such a shader takes advantage of the assumption of a uniform density inside the mesh surface, which makes it faster than rendering an arbitrary density volume. Snow mist and snow particles density volumes are rendered using Houdini Mantra™ standard smoke shader *Billowy Smoke*. The density of the snow particles volume is scaled by a factor of 100 at render time to match the snow opacity from our observations.

By relying on state of the art commercial software, our implementation demonstrates that the proposed approach is easy to integrate into current production pipeline tools. While we used Houdini™ in our implementation, the same base simulation paradigms are implemented in most popular commercial software. Therefore, our approach is not limited to a specific software vendor.

3.7 Results

We have tested our approach on a wide variety of scenarios. The animations for the snow imprint examples are available in the accompanying video of our paper (Dagenais *et al.*, 2016). In Fig. 3.8, the Stanford bunny is translated down and up in the snow creating an imprint. The contour of the bunny is well captured in the base layer level set. Furthermore, the noise applied on the base layer provides a more natural look at a negligible computation expense. This scenario has been computed with no compression in Fig. 3.8a, and with a high level of compression in Fig. 3.8b. Fewer particles are found in the imprints in the latter case, resulting in a deeper hole. In Fig. 3.9, the Stanford bunny pushes snow as it moves horizontally. Snow is transferred from the base layer to snow particles, and creates a pile of snow in front and beside of the bunny's path. These particles add the dynamics of snow not modeled by the base layer. As can be seen in Fig. 3.9b, a smaller pile of snow is formed when using a higher compression ratio. Similarly, a sleigh leaves tracks in the snow in Fig. 3.10. Snow pushed by the sleigh shows a very high level of details. Our approach allows such level of details, even with a large simulation domain, by adding snow particles only where details are needed.

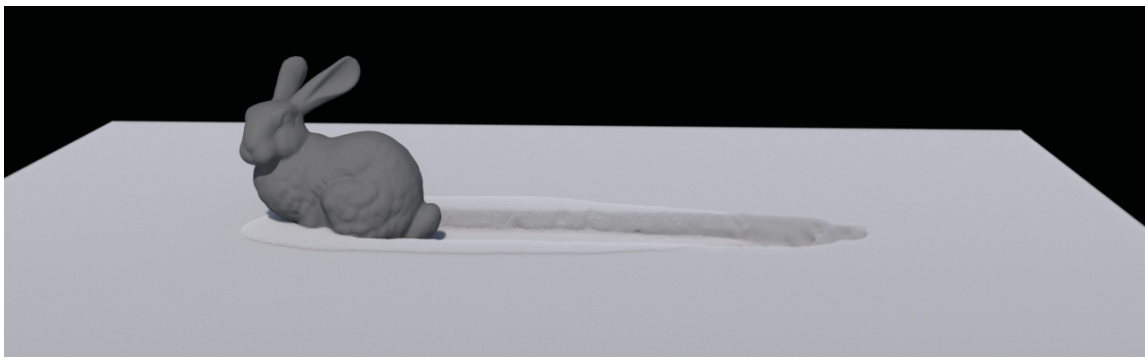
Our approach has also been validated with more complex animations, closer to what is found in an animation movie. In Fig. 3.11, a human character walks in the snow. As its feet move upward, snow particles are projected in the air, along with some snow mist. A gorilla walks, spins, and rolls in the snow in Fig. 3.12. All of its interactions with the snow are well captured by our approach. As snow particles are projected in the air, snow mist is automatically added to the fluid simulation, improving the realism, compared to methods relying only on height maps or granular material simulations.



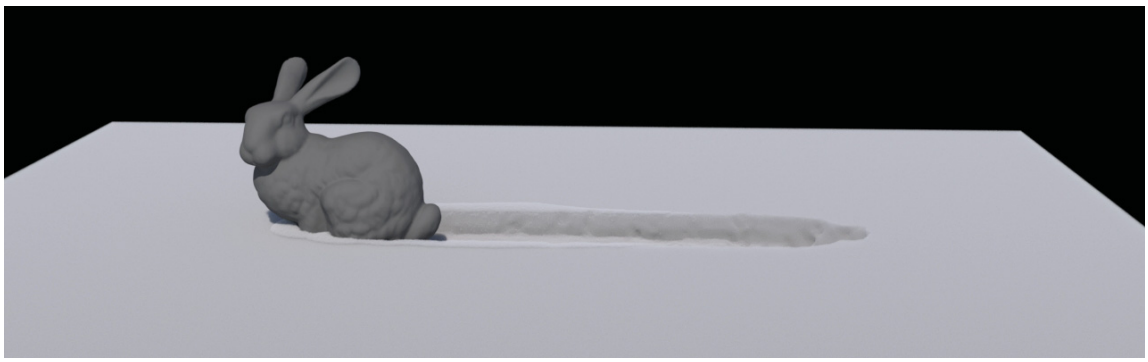
a) $r_c = 0.0$ b) $r_c = 0.75$

Figure 3.8 Stanford bunny imprints. This scenario has been computed with no compression, i.e., with the compression ratio value $r_c = 0.0$ (a) and a high compression ratio $r_c = 0.75$ (b)

The timings for all the snow imprint examples are available in Table 3.1. Most of the time is spent in the snow particles simulation. The base layer is by far the fastest step in our approach. Thus, in production, it can be used to have an early feedback of the imprints in the snow, which can be used to adjust the animations quickly. We were able to lower considerably the computation time for the snow particles simulation by excluding inactive particles during computations (see Fig 3.13). Without such optimization, the dancing gorilla example required an average of $62.20s$ per frame to simulate snow particles. With the optimization, it only required an average of $39.25s$ per frame, a 1.58X speedup. All of our simulations were computed on a 6 CPUs Intel Core i7 with 56 GB of memory. The gorilla model was obtained from the TOSCA (Bronstein, Bronstein & Kimmel, 2008) project, and animated in Autodesk Maya® using a provided example animation setup.



a) $r_c = 0.0$



b) $r_c = 0.75$

Figure 3.9 The Stanford bunny slides to the left, pushing snow particles as it moves. This scenario was computed with no compression (a) and with a high level of compression (b)

Table 3.1 Timings for all snow imprints examples

					Average time per frame (s)			
Example	**Base layer cell size (m)**	**Snow particle radius (m)**	**# snow particles**	**Snow mist cell size (m)**	**Base layer**	**Snow particles**	**Snow mist**	**Total**
Fig. 3.8a	0.005	0.00075	243,653	0.005	1.84	11.64	6.00	19.48
Fig. 3.8b	0.005	0.00075	60,759	0.005	1.84	4.36	5.09	11.29
Fig. 3.9a	0.005	0.00075	1,575,948	0.005	1.68	81.26	9.18	92.12
Fig. 3.9b	0.005	0.00075	396,272	0.005	1.73	15.90	6.18	23.81
Fig. 3.10	0.01	0.003	1,684,770	0.015	1.92	33.73	2.85	38.50
Fig. 3.11	0.0075	0.0025	879,062	0.01	3.15	11.30	6.65	21.10
Fig. 3.12	0.0075	0.0025	2,850,936	0.01	4.59	39.25	13.62	57.46

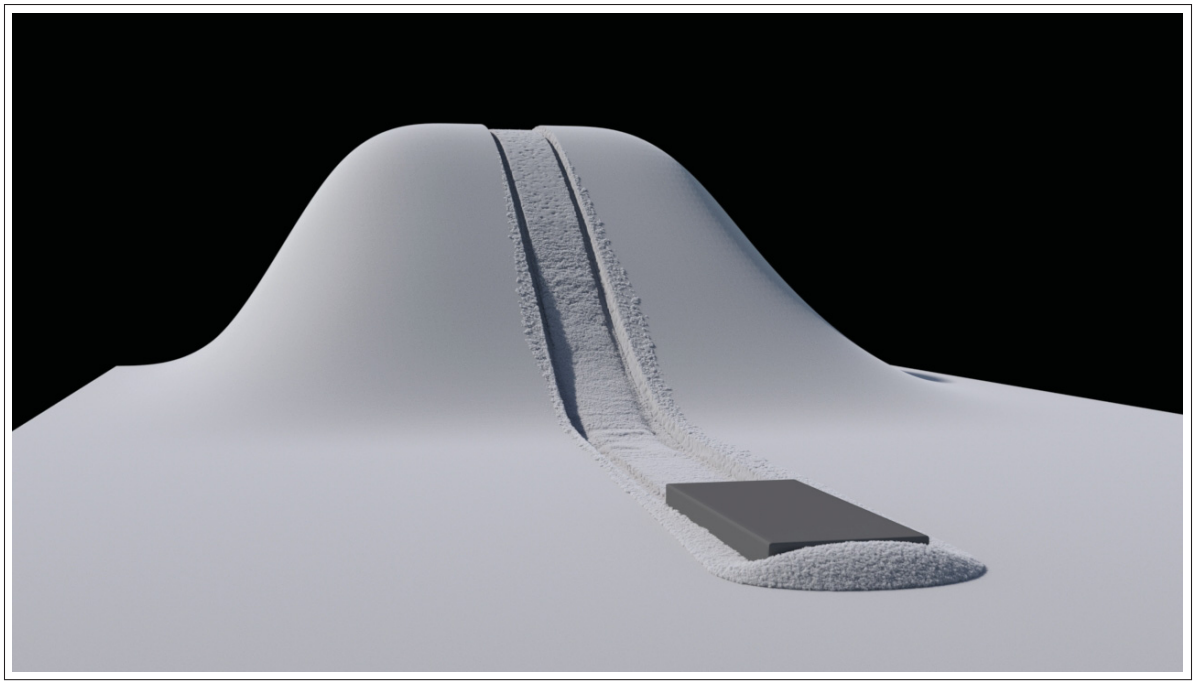


Figure 3.10 A sleigh leaves tracks and pushes snow

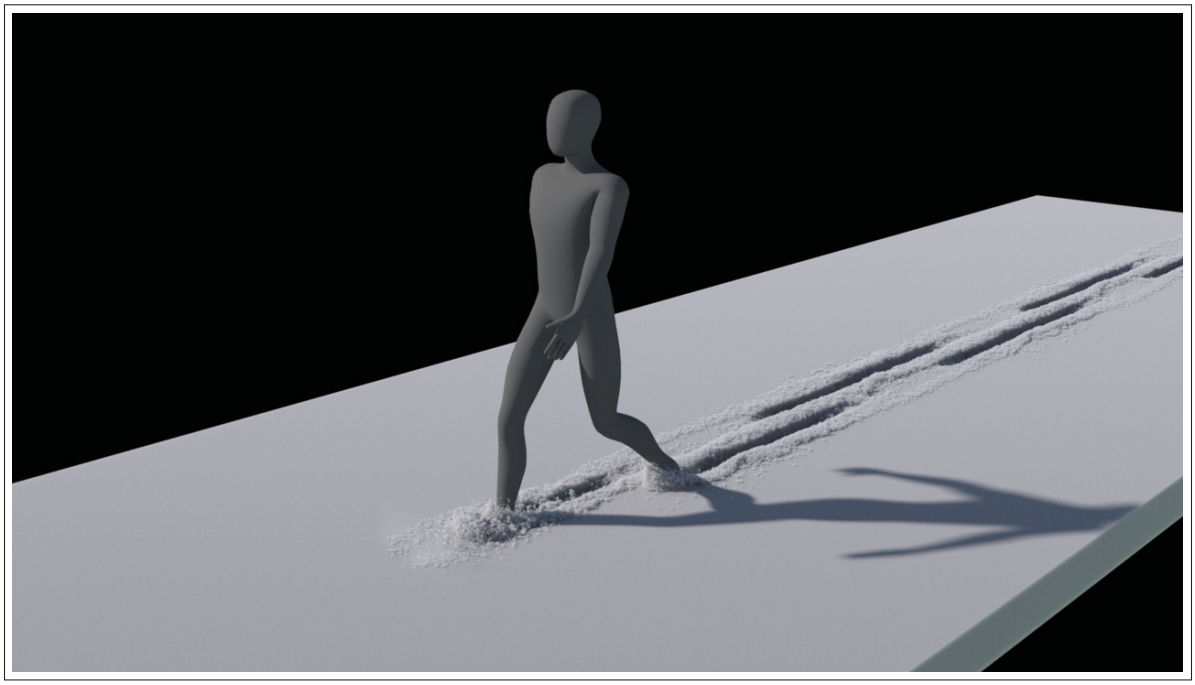


Figure 3.11 A human character walking in the snow

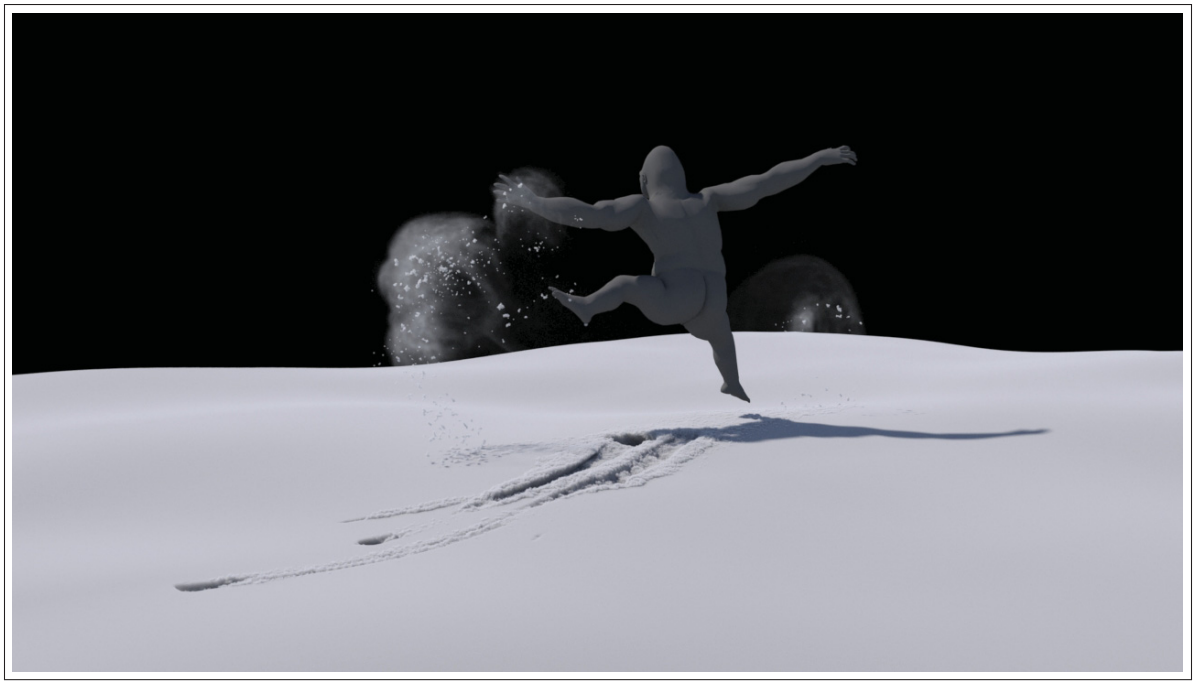


Figure 3.12 A gorilla dances in the snow, leaving imprints as well as projecting snow and snow mist in the air

Our approach considerably lowers memory consumption and computation times by using a level set, i.e., the base layer, to model snow that did not interact with the dynamic objects yet. We compared the memory consumption and timings of our approach with a particles-only simulation, where the whole snow volume was filled with particles, in the walking human character example. Because of memory limitations, a coarser resolution was used, setting the particles radius to $2r$. Even with the coarser resolution, the simulation required 6.9 GB of memory. Using a finer resolution, i.e., with the particles radius set to r, it would require approximately eight times more memory (55 GB). In contrast, the finer resolution simulation computed with our approach fits within 3.5 GB of memory for the base layer and the snow particles. As for the timings, the coarse particles-only simulation required on average $273.1s$ per frame. Using our inactive particles optimization with the coarse particles-only simulation, it could be lowered to $51.86s$ per frame. Still, even with the eight times finer resolution, our full approach required on average only $21.1s$ per frame, which also includes the base layer and snow mist computation times.

Figure 3.13 Only active snow particles (in red), in the areas around dynamic objects and moving particles, are computed in our examples. This considerably lowers computation times

Our approach has some limitations. The coupling between the snow and the dynamic object is unidirectional: the animated objects influence the snow, but the snow has no influence on the objects. Physically-based animation with locomotion controllers cannot be used with our method as it does not model the contact from the snow to the dynamic objects. Nevertheless, this is quite an advantage for typical visual effect animations, as it provides a perfect and predictable animation of the dynamic objects or characters. Even though we can capture a wide range of interactions, some cannot be captured. For example, as our approach assumes that snow remains static until it collides with an object, it is possible to design animations creating sharp snow cliffs and tunnels into which the surrounding base layer snow will never fall.

3.8 Discussion

In this chapter, we introduced an efficient workflow that handles the interaction of snow with dynamic objects. We decomposed the snow into three components that can be controlled

individually. Firstly, the base layer can be controlled to influence the imprints' shape and details, at a low computational cost. Secondly, the snow particles control the complex dynamics of snow that gets pushed and thrown in the air. Finally, the snow mist provides control over the thinner airborne snow that behaves according to its surrounding air medium. Our approach lowers memory consumption and computation times by using the most expensive simulations only where needed. Additionally, the surface representation of the base layer allows the use of the same shader on the simulated snow as well as any background snow that would be left out of the simulation for efficiency reasons. Our approach was developed as part of a collaboration with a VFX studio called *Mokko*. Throughout the project, several of their artists and a VFX supervisor validated our results to attest that they are visually plausible and match their need for their ongoing productions.

In the future, we could improve our model so that the base layer snow is transformed to snow particles in other situations such as along sharp slopes or when a dynamic object moves below the base layer surface. Another avenue for future work would be to improve the way compression is handled using a physically-based model that could consider properties such as air temperature, snow temperature, and snow wetness. We could also create a model that scales the drag forces according to the density of snow particles to better mimic the friction of snow with the air. Also, our approach could be modified to convert back static snow particles to the base layer. We believe that such an improvement could lower the memory consumption. However, care must be taken to avoid sudden jumps or flickering of the surface. A possible solution would be to convert only hidden layers of particles, i.e., particles that are fully covered, similarly to Zhu & Yang (2010) and Wong *et al.* (2015). Nevertheless, it should be noted that such a modification would prevent the user from running the entire base layer simulation before simulating snow particles. Finally, we could experiment with the some more recent work on swept volumes (Sellán, Aigerman & Jacobson, 2021) to compute the volume that covers the full range of motion of our dynamic objects. It could be used to generate a dent-free swept volume without requiring our smoothing step, providing a volume that better follows the contour of our dynamic objects. However, this technique requires as inputs a solid shape with a trajectory

function that describes a rigid motion. It would not be trivial to apply it to deformable shapes such as an animated character.

CHAPTER 4

REAL-TIME FLUID SIMULATION

The work described in this chapter has been presented at *VRIPHYS 2018* (Dagenais *et al.*, 2018b), where it won the best paper award:

François Dagenais, Julián Guzmán, Valentin Vervondel, Alexander Hay, Sébastien Delorme, David Mould, and Eric Paquette. *Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water*, VRIPHYS 2018 Workshop on Virtual Reality Interaction and Physical Simulation, Delft, The Netherlands, April 15-16, 2018.

An extended version has later been published in the *Computers & Graphics* journal (Dagenais *et al.*, 2018a):

François Dagenais, Julián Guzmán, Valentin Vervondel, Alexander Hay, Sébastien Delorme, David Mould, and Eric Paquette. *Extended virtual pipes for the stable and real-time simulation of small-scale shallow water*, Computers & Graphics, Vol. 76, pages 84-95, Nov. 2018.

In these publications, three students were directly involved in the developpement of the described approach. My work focused on the simulation itself, the novel viscosity model, the surface construction, and the GPU implementation of such aspects of the paper. The other students contributed mainly to the real-time meniscus model and the extended pipes to approximate the liquid flow in fully-flooded cells. Their contribution won't be described in this thesis, see the publication for more details about these aspects.

In this chapter, we focus on real-time simulation of shallow water at small scales, such as in scenarios of spilled coffee or bleeding during surgery (Fig. 4.1). In such situations, thin layers of liquid flow on a surface. At this scale, effects such as the viscous drag force exerted on the liquid by the surface of the obstacles are much more prominent.

In many real-time contexts, such as those involving medical applications and games, it is necessary to have a very efficient simulation since other systems are running on the same

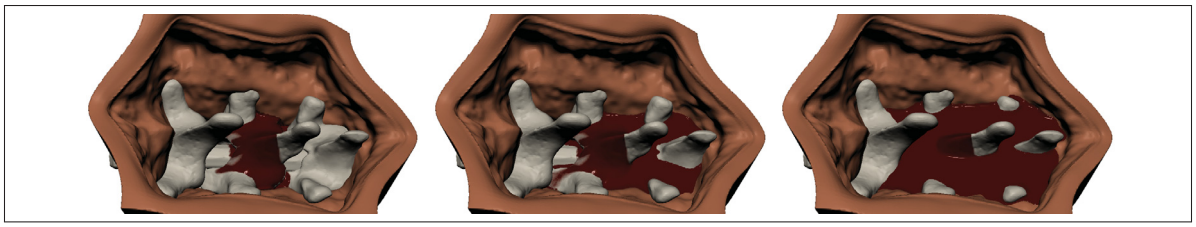


Figure 4.1 Frames from an animation where blood flows from a vertebra

resources. For example, a virtual surgery simulation might need to process user inputs, provide haptic feedback, and even run a soft-body simulation (Boukadida, Andrews & Paquette, 2019) within a few milliseconds. A full 3D simulation is often too expensive; only very coarse resolutions can be achieved in real time. However, a coarse 3D resolution fails to represent thin films of liquids, such as blood or paint flowing over a surface. To reduce computation times, several methods focus on performing a 2D simulation on a heightmap; these methods include those of Chentanez & Müller (2010), and Mei, Decaudin & Hu (2007). Such methods can simulate liquids that are arbitrarily deep or shallow, with no impact on the resolution of the simulation.

Our approach builds upon the VP method (Šťava, Beneš, Brisbin & Křivánek, 2008). Previous papers do not use a physical model to handle the viscous drag force from the terrain. This force is non-negligible for various liquids such as blood and paint. To allow more complex terrain geometries that contain overhangs and holes, previous researchers extended virtual pipe methods to use a multi-layered heightmap (Borgeat *et al.*, 2011; Kellomäki, 2014) and created interconnections between the layers to allow the liquid to flow between them. Such configurations are important for different scenarios, such as when blood should flow below organs in a surgery simulation. However, current multi-layered VP methods have a limited surface representation, with some leading to discontinuities in the surface mesh, while others are unable to accommodate multiple overlapping layers.

Our approach extends the previous work, enhancing both the behavior and the surface representation. The behavior is improved using a physically-based viscosity model that

is both fast and stable. As for the surface representation, our improved multi-layered surface reconstruction does not suffer from discontinuity issues. Our surface optimization approach provides a correction to the mesh surface, preventing interpenetrations with the underlying terrain geometry. We work on the simulation of moderate amounts of liquid, roughly in the 10 ml to slightly over 1 liter range. To summarize, our contributions are as follows:

- We present a physically-based stable viscosity model and analyze its behavior compared to an offline simulation.
- We perform independent multi-layered surface reconstruction with smooth boundaries.
- We use a surface optimization to prevent unwanted interpenetrations between the heightmap and an arbitrary surface.

4.1 Liquid Simulation

Our approach targets the real-time simulation of small amounts of liquid based on the VP method (Sec. 4.1.1). Our contribution can be divided into two categories: improving the fluid behavior and improving the surface reconstruction. On the behavior side, we first propose a new formulation for the viscosity (Sec. 4.1.2), allowing a varying amount of viscosity and widening the range of behavior from water to liquids such as blood and paint. Our surface improvements include a reconstruction with smooth boundaries that adapts its topology to match that of the multi-layered simulation. Furthermore, a surface optimization prevents various forms of artifacts (Sec. 4.2).

4.1.1 Virtual Pipes Simulation

This section explains how we combined different VP methods to derive our specific VP model. Our base liquid simulation follows the VP formulation introduced by O'Brien & Hodgins (1995), with the multi-layer structure of Kellomäki (2014). The VP method uses a 2D simulation grid divided into 2D cells. In turn, each cell has one or multiple columns when we extend to the multi-layered VP model. Each column corresponds to a range $[\min_i, \max_i]$ along the vertical axis, with base height b_i, liquid height h_i, and maximum range $\max_i$ (Fig. 4.2). Our terrain

data comes from a 3D signed distance field representing solid objects. We choose Δx for our simulation grid to be the same size as the terrain cell size so that we capture all of the details from the terrain. At initialization and as needed when the terrain changes, the column properties concerning the terrain ($\min_i$, b_i, and $\max_i$) are set by ray casting the terrain distance field upward through the center of a cell.

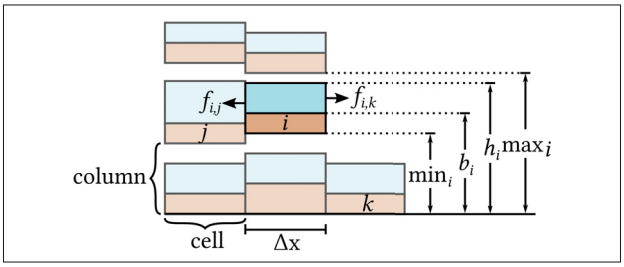


Figure 4.2 The simulation grid is divided into cells, which in turn contain one or more columns. Here we see three linked columns i, j, and k, and column i's properties: its range $[\min_i, \max_i]$ along the vertical axis, its base height b_i, and its liquid height h_i

Adjacent columns are connected using virtual pipes through which they exchange liquid. In the multi-layer case, a pipe connects two adjacent columns i and j when their liquid ranges $[b_i, \max_i]$ overlap. Note that one column can be connected to multiple columns from the same adjacent cell. As in the work of Mei *et al.* (2007), pipe connections are created for the 4-neighborhood. These connections are created at the beginning of the simulation and adjusted when there is a change in the geometry of the obstacles. Throughout this chapter, we often use the term *terrain* to refer to any obstacle (soft or rigid) lying below the columns of liquid. At each simulation step, the flux $f_{i,j}$ between a column i and its neighbor column j is updated using their difference in hydrostatic pressure and an explicit Euler integration:

$$f_{i,j}^{t+\Delta t} = \zeta f_{i,j}^{t} + \Delta t A \frac{g(h_i^t - h_j^t)}{l}, \tag{4.1}$$

where ζ is the friction parameter introduced by Mould & Yang (1997), Δt is the simulation timestep, g is the gravitational acceleration, A is the cross-section area of the virtual pipe, and l is the length of the virtual pipe. As in the work of Št'ava *et al.* (2008), we set l to be equal

to the grid cell size Δx, and $A = \Delta x^2$. Instead of adding a friction force in the simulation, the damping is added in the simulation by scaling the amount of flux from the previous timestep that is preserved using the friction parameter ζ. This approach has the advantage of being stable for $\zeta \in [0, 1]$. We want ζ to have the same effect irrespective of the time step. As such, we propose to set $\zeta = \omega^{\Delta t}$, where $\omega = [0, 1]$ is the fraction of flux retained per unit time. In our examples, we set $\omega = 0.5$. Note that $f_{i,j} = -f_{j,i}$; thus we compute the flux once per pipe. In order to prevent the liquid height from dropping below the base height, the outflow fluxes, i.e, $f_{i,j}^{t+\Delta t} > 0$, are scaled by a factor K_i^{min} as described by Mei *et al.* (2007):

$$K_i^{min} = \min\left(1, \frac{(h_i - b_i)A}{\Delta t \sum_{j \in neighbor(i)} \max(0, f_{i,j}^{t+\Delta t} : w)}\right).$$

Similarly, to prevent the liquid height from going above the maximum height, we scale the inflow fluxes as explained by Kellomäki (2014). This is done by first predicting the overlap $d_{overlap,i}$ between the liquid and the upper column using only the inflow fluxes:

$$d_{overlap,i} = \left(h_i + d_{total,i}\right) - \max_i,$$

where $d_{total,i}$ is the liquid height variation in column i from its inflow fluxes, ignoring the outflow fluxes:

$$d_{total,i} = \frac{\Delta t}{\Delta x^2}\left(\sum_{j \in \text{neighbor}(i)} \max(0, -f_{i,j}^{t+\Delta t})\right).$$

If an overlap is detected, i.e., $d_{overlap,i} > 0$, the inflow fluxes are then scaled by a factor K_i^{max}:

$$K_i^{max} = \frac{d_{total,i} - d_{overlap,i}}{d_{total,i}}.$$

The scaled fluxes are then used to compute the new liquid heights, again using an explicit Euler integration:

$$h_i^{t+\Delta t} = h_i^t + \frac{\Delta t}{\Delta x^2} \sum_{j \in \text{neighbor}(i)} f_{j,i}^{t+\Delta t}. \tag{4.2}$$

The main steps of our simulation loop are shown in Alg. 4.1. The line in bold is the step that we added to the VP method to handle viscosity.

Algorithm 4.1 Real-time shallow liquid simulation loop

```
1 Update liquid fluxes
2 Apply viscosity (Sec. 4.1.2)
3 Scale liquid fluxes
4 Update liquid heights
5 Source liquid in the simulation
```

4.1.2 Viscosity Model

When a thin layer of liquid flows on a surface, it is slowed down by the shear stress forces from its interaction with the terrain. This effect is propagated further away through the liquid by the viscous shear forces. As the depth of the liquid layer increases, the impact of this interaction on the overall liquid velocity decreases, at a rate determined by its viscosity. Handling a wide range of liquid depths is important in many applications, including virtual surgery, for example. Hence, our goal is to handle a wide range of depths in a physically-based manner, which cannot be done by the current VP method.

Our proposed viscosity model is based on the Navier-Stokes equations, which are based on the velocity. The VP method works with the flux $f_{i,j}$ instead of the velocity $u_{i,j}$; These are related as follows:

$$f_{i,j} = \frac{u_{i,j}}{C}, \tag{4.3}$$

where $C = \Delta x(h - b)$ is the cross-sectional area of the liquid's flow from one column to its neighbor. Note that $u_{i,j}$ is a scalar that represents the speed of the flow along the pipe orientation. For simplicity we assume that the vertical axis is z; pipes are thus aligned with either the x- or y-axis. Because $u_{i,j}$ lies between adjacent columns, it can be interpreted as a staggered grid (see Fig. 4.3), where $u_{i,j} = u_x$ or u_y, depending on its orientation.

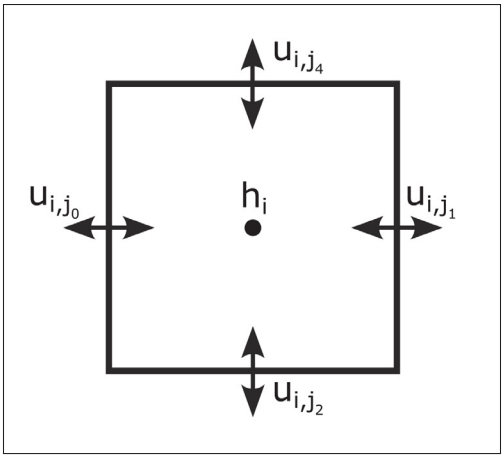


Figure 4.3 In a staggered grid, the velocity is stored at the center of the faces between adjacent cells

The VP method assumes that the liquid's properties are constant along the vertical axis. However, because viscous forces are computed from the spatial differences of the velocity, this assumption would result in no shear viscous forces along the vertical axis. For that reason, we define a velocity profile that varies vertically $u^p_{i,j}(z)$, and interpret $u_{i,j}$ as the average vertical velocity. We derive the relationship between $u^p_{i,j}(z)$ and $u_{i,j}$ in Sec. 4.1.2.1, which leads to:

$$u^p_{i,j}(z) = -\frac{3u_{i,j}}{H^2}\left(\frac{z^2}{2} - Hz\right), \tag{4.4}$$

where H is the liquid's depth $(h_i - b_i)$ of the column from which the flux originates. Using this vertically varying velocity, we derive in Sec. 4.1.2.2 the effect of the viscosity on the average velocity $u_{i,j}$:

$$u^{n+1}_{i,j} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) u^n_{i,j},$$

where ν is the kinematic viscosity of the liquid. Using Eq. 4.3, we can convert from velocity to flux:

$$f^{n+1}_{i,j} C^{n+1} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) f^n_{i,j} C^n.$$

By assuming that the liquid depth remains the same during the timestep, we get $C^n = C^{n+1}$:

$$f_{i,j}^{n+1} = \left(\frac{H^2}{H^2 + 3\Delta t\nu}\right) f_{i,j}^{n}. \tag{4.5}$$

At each step, this equation is used to enforce viscosity on the fluxes computed from Eq. 4.1. The following sections demonstrate how this viscosity model is derived.

4.1.2.1 Velocity Profile

In this section, we derive the vertically varying velocity $u_{i,j}^{p}(z)$. While an arbitrary function with an average value of $u_{i,j}$ could be used, we choose a physically-based one that will better approximate a real-life flow. For simplicity, we assume that the base of the liquid is at $z = 0$ and that the top is at $z = H$. The function $u_{i,j}^{p}(z)$ is thus defined inside the liquid, i.e., for $0 \leq z \leq H$. The velocity is derived from the incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nu \nabla \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \mathbf{g}, \tag{4.6}$$

$$\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0, \tag{4.7}$$

where p is the pressure, $\mathbf{g} = (0, 0, g_z)$ is the gravity, and $\mathbf{u} = (u_x, u_y, u_z)$ is the velocity. To make the velocity profile a function of only the height z inside the liquid, we show that the Navier-Stokes momentum equation (Eq. 4.6) can be simplified. To begin, the first term can be removed by considering a steady flow:

$$\frac{\partial \mathbf{u}}{\partial t} = 0. \tag{4.8}$$

We also assume the velocity is constant along the horizontal plane:

$$\frac{\partial \mathbf{u}}{\partial x} = \frac{\partial \mathbf{u}}{\partial y} = \mathbf{0}. \tag{4.9}$$

This assumption allows us to ignore contributions from neighbor cells in order to derive a fast and stable analytical solution (see Sec. 4.1.2.2). From this, Eq. 4.7, which enforces the incompressibility of the liquid, yields:

$$\frac{\partial u_z}{\partial z} = 0. \tag{4.10}$$

This result tells us that the velocity's vertical component cannot vary vertically when its horizontal components are constant, otherwise the liquid would lose or gain volume. At the terrain level, we use a no-slip boundary condition, and at the surface level, a traction-free boundary condition:

$$\mathbf{u}|_{z=0} = \mathbf{0}, \left.\frac{\partial \mathbf{u}}{\partial z}\right|_{z=H} = \mathbf{0}. \tag{4.11}$$

The terrain boundary condition and Eq. 4.10 tell us that $u_z = 0$ everywhere, and can thus be ignored. Using this result and Eq. 4.9, the second term of the Navier-stokes momentum equation becomes zero:

$$\begin{aligned} \mathbf{u} \cdot \nabla \mathbf{u} &= u_x \frac{\partial \mathbf{u}}{\partial x} + u_y \frac{\partial \mathbf{u}}{\partial y} + u_z \frac{\partial \mathbf{u}}{\partial z} && (4.12) \\ &= u_x(0) + u_y(0) + (0)\frac{\partial \mathbf{u}}{\partial z} = \mathbf{0}. && (4.13) \end{aligned}$$

Finally, the third term can be simplified similarly:

$$-\nu \nabla \cdot \nabla \mathbf{u} = -\nu \left(\frac{\partial^2 \mathbf{u}}{\partial x^2} + \frac{\partial^2 \mathbf{u}}{\partial y^2} + \frac{\partial^2 \mathbf{u}}{\partial z^2} \right). \tag{4.14}$$

Using Eq. 4.9, the first two terms become zero, yielding:

$$-\nu \nabla \cdot \nabla \mathbf{u} = -\nu \frac{\partial^2 \mathbf{u}}{\partial z^2}. \tag{4.15}$$

After these simplifications, Eq. 4.6 now becomes:

$$-\nu \frac{\partial^2 \mathbf{u}}{\partial z^2} = -\frac{1}{\rho} \nabla p + \mathbf{g}. \tag{4.16}$$

Now that the Navier-Stokes equations have been simplified, we can derive the horizontal velocity as a function of z. As we use fluxes between cells, we derive the velocity along the u_x and u_y axes. We will show the derivations for u_x, and u_y can be obtained in the same way. From Eq. 4.16, considering the x component, we get:

$$\nu \frac{\partial^2 u_x}{\partial z^2} = \frac{1}{\rho} \frac{\partial p}{\partial x}. \tag{4.17}$$

We assume the horizontal pressure variation to be constant, and replace it by a constant w:

$$\nu \frac{\partial^2 u_x}{\partial z^2} = w, \tag{4.18}$$

Integrating both sides of the equality with respect to z, and also dividing both sides by ν yields:

$$\frac{\partial u_x}{\partial z} = \frac{wz + D}{\nu}. \tag{4.19}$$

Using the surface traction-free boundary condition from Eq. 4.11 to find the value of D yields:

$$\frac{\partial u_x}{\partial z} = \frac{wz - wH}{\nu}. \tag{4.20}$$

Integrating both sides of the equality with respect to z a second time, and rearranging the terms yields:

$$u_x = \frac{w}{\nu} \left(\frac{z^2}{2} - Hz \right) + E. \tag{4.21}$$

Using the ground no-slip boundary condition from Eq. 4.11, we find that $E = 0$:

$$u_x = \frac{w}{\nu} \left(\frac{z^2}{2} - Hz \right). \tag{4.22}$$

To link the velocity function to the flux velocity $u_{i,j}$, we find the expression of the pressure gradient w that will make the average velocity of u_x inside the liquid equal to $u_{i,j}$:

$$u_{i,j} = \frac{1}{H}\int_0^H \frac{w}{\nu}\left(\frac{z^2}{2} - Hz\right) dz = -\frac{wH^2}{3\nu}, \tag{4.23}$$

$$w = -\frac{3\nu u_{i,j}}{H^2}. \tag{4.24}$$

Using that result in Eq. 4.22 yields:

$$u_x = -\frac{3u_{i,j}}{H^2}\left(\frac{z^2}{2} - Hz\right). \tag{4.25}$$

Performing the same derivation on u_y yields the same result. Knowing that a flux is always aligned with the $x-$ or $y-$ axis, u_x can be substituted by $u^p_{i,j}(z)$:

$$u^p_{i,j}(z) = -\frac{3u_{i,j}}{H^2}\left(\frac{z^2}{2} - Hz\right). \tag{4.26}$$

This result is used as our velocity profile function in this chapter. As required, its average value is $u_{i,j}$ inside the interval $[0, H]$. A graph of this function is shown in Fig 4.4.

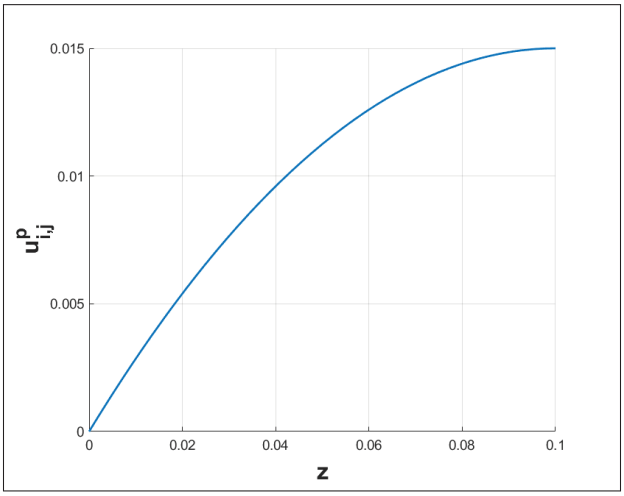


Figure 4.4 A graph showing the velocity profile function $u^p_{i,j}(z)$ for $u_{i,j} = 0.05$ and $H = 0.1$

4.1.2.2 Viscosity Derivation

Now that the vertically varying velocity $u^p_{i,j}(z)$ (Sec. 4.1.2.1) has been defined, we can derive the viscosity model. Our goal is to determine how to update the velocity, i.e., $\frac{\partial u_{i,j}}{\partial t}$, based on the viscosity. Our model accounts for the effect of viscosity from the terrain to the liquid surface, ignoring the effect of neighbor cells. While this is less accurate, neglecting the relation between neighbor cells allows us to implicitly integrate each column independently, instead of having to solve a large system of equations. We derive our viscosity model by considering only the viscous term of the Navier-Stokes momentum equation. Here, we show the derivation for $\frac{\partial u_x}{\partial t}$, but the same process can be applied to $\frac{\partial u_y}{\partial t}$:

$$\frac{\partial u_x}{\partial t} = \nu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \right). \tag{4.27}$$

With the assumption of Eq. 4.9, Eq. 4.27 is simplified to:

$$\frac{\partial u_x}{\partial t} = \nu \frac{\partial^2 u_x}{\partial z^2} \tag{4.28}$$

Using the velocity profile defined by Eq. 4.26, this equation becomes:

$$\frac{\partial u_x}{\partial t} = \nu \frac{\partial^2}{\partial z^2} \left[-\frac{3u_{i,j}}{H^2} \left(\frac{z^2}{2} - Hz \right) \right] = -\frac{3\nu u_{i,j}}{H^2} \tag{4.29}$$

This result is used to compute how the velocity $u_{i,j}$ varies over time:

$$\frac{\partial u_{i,j}}{\partial t} = \frac{\partial}{\partial t} \left(\frac{1}{H} \int_0^H u_x dz \right). \tag{4.30}$$

Using the Leibniz integral rule, and simplifying the problem by assuming that the liquid depth H does not vary over time, yields:

$$\frac{\partial u_{i,j}}{\partial t} = \frac{1}{H}\int_0^H \frac{\partial u_x}{\partial t} dz \tag{4.31}$$

$$= \frac{1}{H}\int_0^H \left(-\frac{3\nu u_{i,j}}{H^2}\right) dz \tag{4.32}$$

$$= -\frac{3\nu u_{i,j}}{H^2}. \tag{4.33}$$

Finally, we integrate the average velocity implicitly using the backward Euler method to get the temporal update of the velocity, considering the viscosity:

$$u_{i,j}^{n+1} = u_{i,j}^{n} - \Delta t \frac{\partial u_{i,j}}{\partial t} \tag{4.34}$$

$$= u_{i,j}^{n} - \frac{3\Delta t \nu}{H^2} u_{i,j}^{n+1}. \tag{4.35}$$

Rearranging the terms yields:

$$u_{i,j}^{n+1} = \left(\frac{H^2}{H^2 + 3\Delta t \nu}\right) u_{i,j}^{n}. \tag{4.36}$$

This final result is used to approximate the effect of viscosity in our simulations. The right-hand side coefficient is guaranteed to be in the $[0, 1]$ range for $0 \leq 3\Delta t\nu$ and $0 \leq H$. This means that no energy is added in the simulation, and thus it remains stable even for arbitrarily small or large kinematic viscosity values. Fig. 4.5 shows how the coefficient of Eq. 4.5 varies for a representative range of values of H and ν.

The behavior can be roughly characterized as follows. As the liquid depth H increases, the coefficient of the right-hand side gets closer to one, lessening the impact of viscosity. Similarly, the viscosity ν damps the fluxes more severely as H becomes smaller. This shows that our viscosity model is stable and produces the intended behavior. Furthermore, it is simple and fast to compute, making it ideal for real-time purposes.

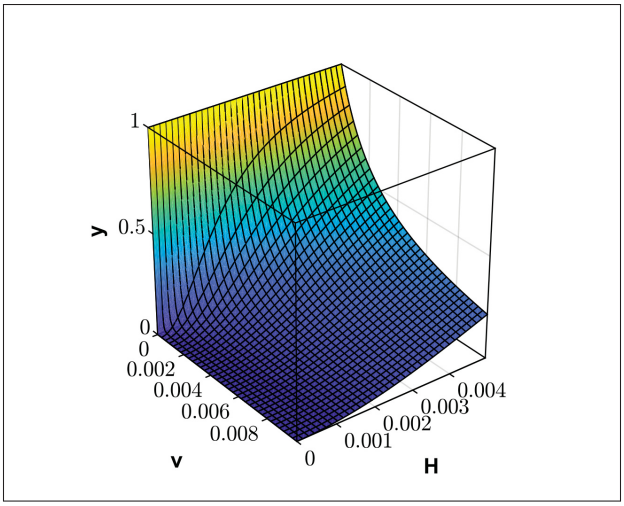


Figure 4.5 A 3D graph showing how the viscosity coefficient $y = \frac{H^2}{H^2+3\Delta t\nu}$ varies for $\nu \in [0, 0.1]$ and $H \in [0, 0.005]$

4.2 Liquid Surface

There are several requirements for the liquid surface: real-time generation and rendering, support for multi-layer surfaces, and flexibility to use an arbitrary geometry for the obstacles' surface. To render the liquid surface, Borgeat *et al.* (2011) displace the terrain using the liquid height in the nearest column, and adjust the vertex colors and normals. This is problematic as the meshes of the obstacles are unlikely to have the appropriate resolution and uniformity to correctly represent the liquid. Furthermore, discontinuities are introduced at overhangs where the obstacle meshes of both levels are not connected to each other. Fig. 4.6 and the paper's accompanying video show such problems on the surface near fully submerged overhangs. Kellomäki (2014) displaces the vertices of a regular mesh grid based on the height of the topmost columns, but this method assumes a single continuous surface over the whole simulation domain, which is not the case in multi-layer scenarios. Considering the limitations of current methods, we devised a new surface creation approach which detects links between adjacent columns (Sec. 4.2.1), and connects them using an optimal triangle configuration (Sec. 4.2.2). Furthermore, we handle boundaries of the liquid to ensure a coherent surface (Sec. 4.2.3). Finally, we propose an optimization approach

to prevent conflicting intersections between the liquid surface and the surface of the obstacles (Sec. 4.2.4). Alg. 4.2 presents the steps for the preparation of the surface.

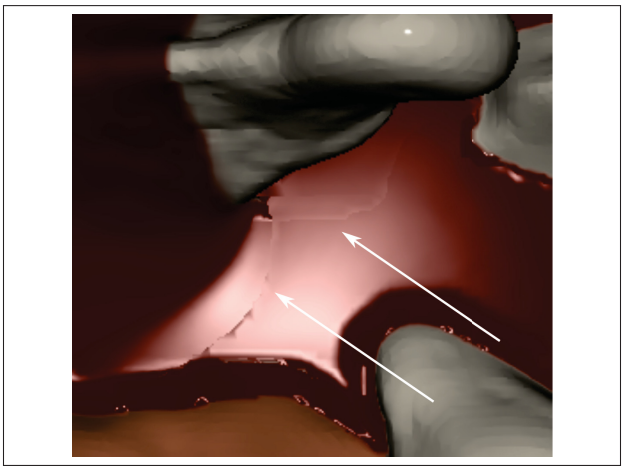


Figure 4.6 Surface seam problems from the method of Borgeat *et al.* (2011)

Algorithm 4.2 Steps related to the preparation of the surface

1. Surface links (Sec. 4.2.1)
2. Surface creation (Sec. 4.2.2)
3. Surface boundaries (Sec. 4.2.3)
4. Surface optimization (Sec. 4.2.4)

4.2.1 Multi-Layered Surface Links

When handling multiple layers, the surface of the liquid gets more complex: adjacent cells could have a different number of columns, and parts of the obstacle geometry could exhibit overhangs. This increases the complexity of the surface creation. It is thus important to derive a robust approach that can handle all cases. In our approach, neighbor columns that will be linked by the liquid mesh surface are identified using a multi-layered link test (Fig. 4.7). With this test, columns are linked if their respective liquid heights fit between each other's minimum and maximum heights. As such, we define two neighbor columns i and j as linked if: $\min_j < h_i < \max_j$, and $\min_i < h_j < \max_i$, as shown in Fig. 4.7. In contrast to the pipe connections, the surface

links are computed for the 8-neighborhood of each cell. They are computed for each pair of adjacent wet or wet–dry columns. A dry column is defined as a column where $h_i \leq b_i$. This approach works in the general case, while a few special cases occurring at the boundary are handled differently (Sec. 4.2.3). The surface links change over the course of the simulation and are verified at each frame (Fig. 4.8). In the next section, we explain our approach to generating the surface from the surface links.

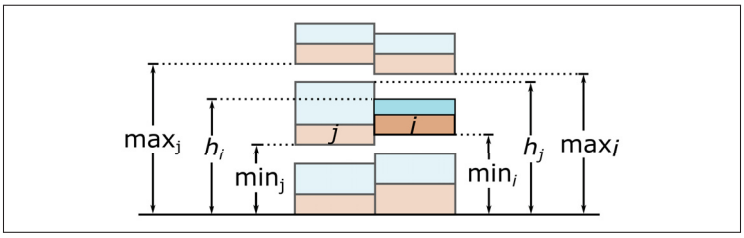


Figure 4.7 Neighbor columns are linked if their liquid height lies within one another's range $[\min, \max]$

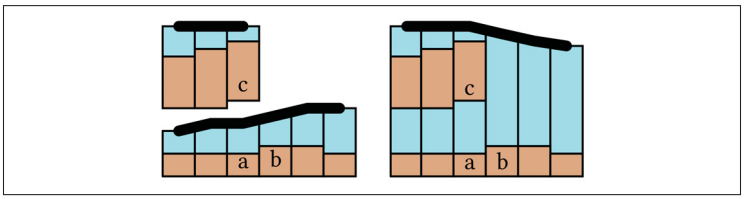


Figure 4.8 Multi-layered linking. The thick lines show linked columns. On the left: columns a and b are initially linked. On the right: as the liquid height of b increases, it becomes linked with c

4.2.2 Surface Creation

Our work extends the idea of displacing the vertices of a regular mesh grid to multi-layered simulations. At the beginning of the simulation, a 3D vertex is allocated for each column, and matches the 2D coordinates of its column. Then, at each frame, its vertical position is set to match the liquid's height. Afterward, these vertices are used to create triangular faces across adjacent linked columns (based on the link test from Sec. 4.2.1). To that end, quartets of neighbor cells $\{(i, j), (i + 1, j), (i, j + 1), (i + 1, j + 1)\}$ are iterated. For each quartet of cells,

we analyze the links between their columns to find their corresponding triangle configuration, as shown in Fig. 4.9. First, we find the groups of four mutually interlinked columns (these will form two triangles). From the remaining columns, we then identify the groups of three mutually interlinked columns (these will form a single triangle). Finally, the remaining groups of two mutually interlinked columns are discarded as they do not correspond to a triangle. For groups of four linked neighbor columns, two triangle configurations are possible. In such cases, we pick the configuration for which the sum of the liquid height of the diagonal endpoints is the greatest, as described by Chentanez & Müller (2010), in order to align the diagonal edge of the triangles with the curvature of the surface.

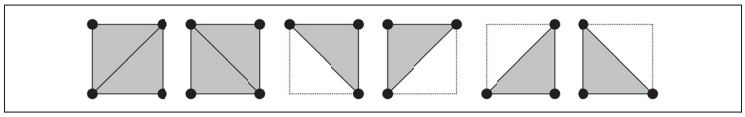


Figure 4.9 Possible triangle configurations between 4- or 3-linked columns inside a quartet. The black points correspond to column centers

Before rendering, the normals are computed from the liquid heights, and the vertex opacity o_i is adjusted to the liquid depth:

$$o_i = \min\left(\frac{h_i - b_i}{\text{depth}_{\text{max}}}, 1\right),$$

where $\text{depth}_{\text{max}}$ is the depth from which the liquid starts being opaque. This enhances realism by better matching the opacity of the simulated liquid, and it improves the look of the wet–dry boundary on flat and convex surfaces.

4.2.3 Boundaries

In order to render a smooth liquid boundary and avoid having cracks at the junction with the terrain, the surface requires special treatment at the boundary of the liquid. Wet columns can be linked with dry columns having a base height (and thus liquid height) significantly higher than the liquid height of the wet column. Blindly creating triangles to the liquid height of dry columns would result in unrealistic slopes on the liquid surface near dry columns (Fig. 4.10,

left). The vertex height of dry columns is thus adjusted to the average height of their linked wet neighbors, and the surface normals of dry columns are adjusted in a similar fashion. With these corrected vertices and normals, the surface is smoother and appears as expected (Fig. 4.10, right). Also, during the triangle generation step, the triangle configuration that allows the ends of the diagonal edge to be both inside or both outside the boundary is prioritized.

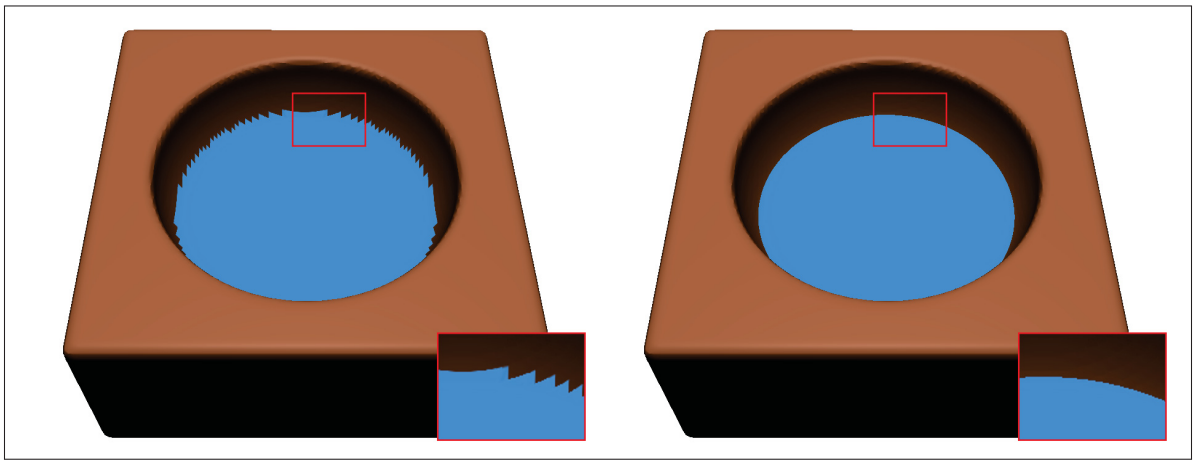


Figure 4.10 Bowl filled with liquid without adjusting boundaries (left), and with boundary adjustments (right)

In some cases, generally below overhangs, the liquid surface in a wet column i might be next to a wall boundary belonging to a neighbor column j to which it is not linked, i.e., $min_j < h_i < max_j$, but $max_i < b_j$. This results in a crack between the wet column and the wall boundary. To prevent such cracks, we create an extra vertex positioned at the center of column j. This new vertex is linked to column i, as well as to its wet neighbor columns that are linked with column i. Its height and normal are then set to the average of these linked neighbor columns. The triangle generation step is triggered anew for all such new boundary links.

4.2.4 Surface Optimization for Rendering

The terrain distance field we get as input can have any orientation, which means that in most cases it is not aligned with the simulation grid. Terrain vertices, resulting from applying the marching-cubes algorithm on the terrain distance field, can thus appear anywhere within a

simulation cell, which often leads to protrusions through the surface of the simulated liquid (Fig. 4.11, left). Such issues are avoided by correcting columns with thin layers of liquid; to prevent surface interpenetrations and z-fighting, any non-zero liquid height is forced to be above a computed minimum height (Fig. 4.11 right). This minimum height $h_k^{\min}$ is a combination of a global parameter to avoid z-fighting and a locally-computed height to avoid interpenetrations. The global parameter is set to $0.05\Delta x$ in our examples. By contrast, the locally-computed height is different for each column: it is precomputed from the terrain mesh, and adjusted when the terrain changes.

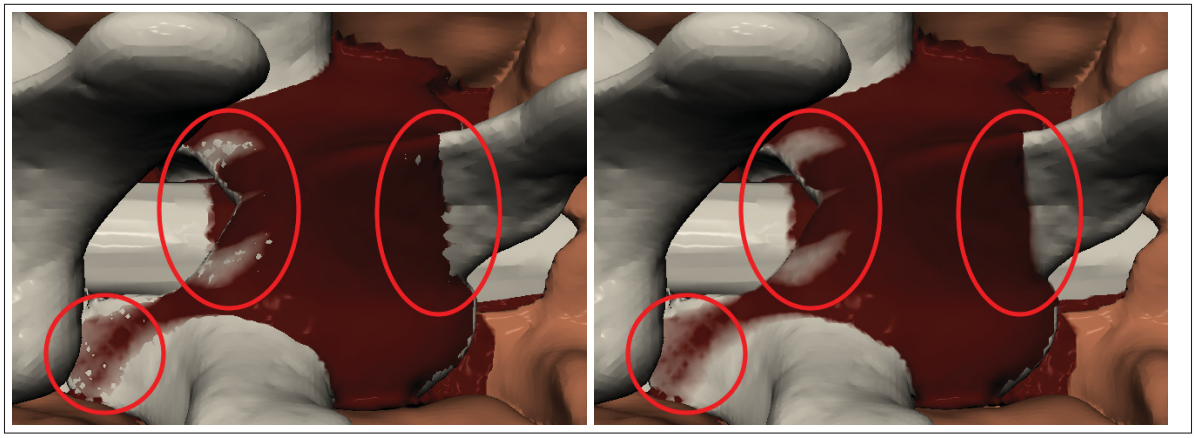


Figure 4.11 A comparison without (left) and with (right) our minimum height correction of the liquid surface heights

The goal of the local height optimization is to compute a minimum height for each column to ensure that all triangles of the liquid surface will be above the terrain mesh vertices, even when the amount of liquid in a column is very small. We iterate through each vertex of the terrain mesh surface, identifying the quartet of four columns surrounding the vertex, and optimizing the local heights to make them as small as possible, under the constraint that the triangles formed by these columns are above the terrain vertex. We express the height of the liquid surface, h_l, along the vertical axis through a terrain vertex as the bilinear interpolation of the heights of the related quartet. The height of each column is the sum of the local height l_k and the base height b_k. We optimize the local heights l_k under the constraint that the bilinear interpolation h_l should be

slightly above the terrain vertex h_v:

$$\min_{l_k} \sum_k \left(w_k l_k^2 \right), \quad \text{subject to} \quad h_v + \epsilon = \underset{k \in \text{quartet}}{\text{bilinear}}(b_k + l_k), \tag{4.37}$$

where w_k are weights assigned to each column, which will be discussed later in this section. This constrained equation is solved analytically using the Lagrange multipliers. As such, the approach is efficient as it does not require that a system of equations be solved for each terrain vertex. The final l_k for a column is the maximum over the l_k computed for all terrain vertices. Once the local heights are computed, the local minimum height $h_k^{\min}$ of each of the four columns can be computed as $h_k^{\min} = b_k + l_k$. The paper's accompanying video shows a visualization of the impact of the position of a terrain vertex on the local minimum heights of a quartet.

The solution of Eq. 4.37 can create unnatural bumps on the surface, generally in regions near a steep slope on the terrain, where the differences between base heights in the quartet are quite large. We prevented such issues by designing weights w_k based on the distance between the column base height b_k and the vertex height h_v:

$$w_k = \begin{cases} \dfrac{\Delta x}{h_v - b_k}, & \text{if } b_k \leq h_v \\ \text{1E10}, & otherwise \end{cases}. \tag{4.38}$$

These weights enforce a greater correction to the columns with a base height at the bottom of a large slope, and limit corrections to those already above the terrain vertex height.

Terrain vertices whose normal is facing downward are skipped. Furthermore, we discard terrain vertices that create a height $h_k^{\min}$ higher than a cell size Δx above the highest base height of the quartet. Such cases usually generate unrealistic bumps near edges of large slopes. Finally, large corrections can be induced by terrain vertices that are extremely close to one of the four columns. To prevent such large corrections, we take the global minimum into account during the calculations of the local minimum height, constraining the l_k to be greater than or equal to the global minimum height.

4.3 Results

We tested our approach with a wide range of scenarios to show its behavior, as well as the impact of its parameters. Results are also available in the accompanying video of our paper (Dagenais *et al.*, 2018a). Parameter values and timings can be found in Table 4.1. We found that dynamic

Table 4.1 Statistics for all the real-time fluid simulation examples: resolution ($N \times N \times Layers$), timestep Δt, cell size Δx, and kinematic viscosity ν as well as maximum/average timings (in ms) per frame for the simulation, surface generation, and rendering

		Δt	Δx	ν	**Simulation**		**Surface**		**Rendering**		**Total**	
Example	**Resolution**	**(ms)**	**(m)**	**(m^2/s)**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**
Surgery (Fig. 4.1)	$200 \times 200 \times 5$	3.00	0.0005	4.0E−6	5.43	4.86	2.11	1.10	0.66	0.34	7.04	6.30
Viscosity (Fig. 4.13a)	$200 \times 200 \times 1$	3.00	0.0005	0	1.05	0.61	0.49	0.43	0.19	0.16	1.63	1.21
Viscosity (Fig. 4.13b)	$200 \times 200 \times 1$	3.00	0.0005	4.0E−6	1.03	0.62	0.52	0.46	0.21	0.16	1.63	1.22
Viscosity (Fig. 4.13c)	$200 \times 200 \times 1$	3.00	0.0005	4.0E−5	1.02	0.60	0.52	0.44	0.25	0.16	2.41	1.21
Viscosity (Fig. 4.13d)	$200 \times 200 \times 1$	3.00	0.0005	4.0E−1	0.93	0.60	0.52	0.41	0.22	0.16	2.40	1.20
Three layers (Fig. 4.15)	$100 \times 100 \times 3$	9.00	0.001	4.0E−6	0.68	0.32	0.85	0.37	0.89	0.17	1.53	0.86

viscosity values in the [1.0E−6, 1.0E−5] range provide realistic results for many liquids such as water, blood, and paint. As with typical fluid simulation methods with an explicit integration, preserving the stability of the simulation requires the timestep to be adjusted according to the Courant–Friedrichs–Lewy (CFL) condition. This can be seen in Fig. 4.12: the timestep is inversely proportional to the number of cells dividing the simulation domain.

An important contribution of our work is the viscosity model for small-scale liquid simulation (Sec. 4.1.2). We tested our approach with varying viscosity values (see Fig. 4.13). The images show the simulation at time $t = 3.0s$, with ν = 0, 4.0E−6, 4.0E−5, and 4.0E−1 m^2/s. As the viscosity increases, the liquid flows more slowly on the inclined terrain. Even with a very large viscosity (Fig. 4.13d), our approach is stable.

We derive our viscosity model by applying some simplifications to the Navier-Stokes equations. One notable simplification is that we neglect the contribution from neighbor cells to the viscosity (4.1.2.2). This means that our model ignores the effect of the viscosity related to the lateral

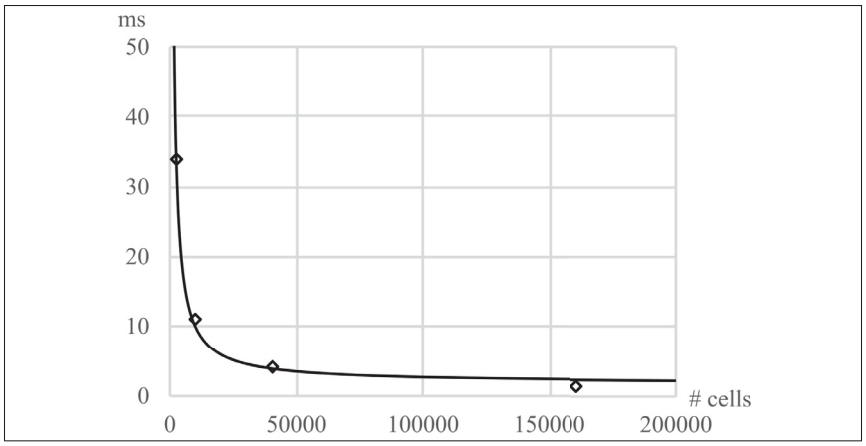


Figure 4.12 Given the same simulation domain, the timestep resulting in a stable simulation is inversely proportional to the number of cells. The figure, as well as its data, have been generated by the co-author of the original paper Julian Guzman

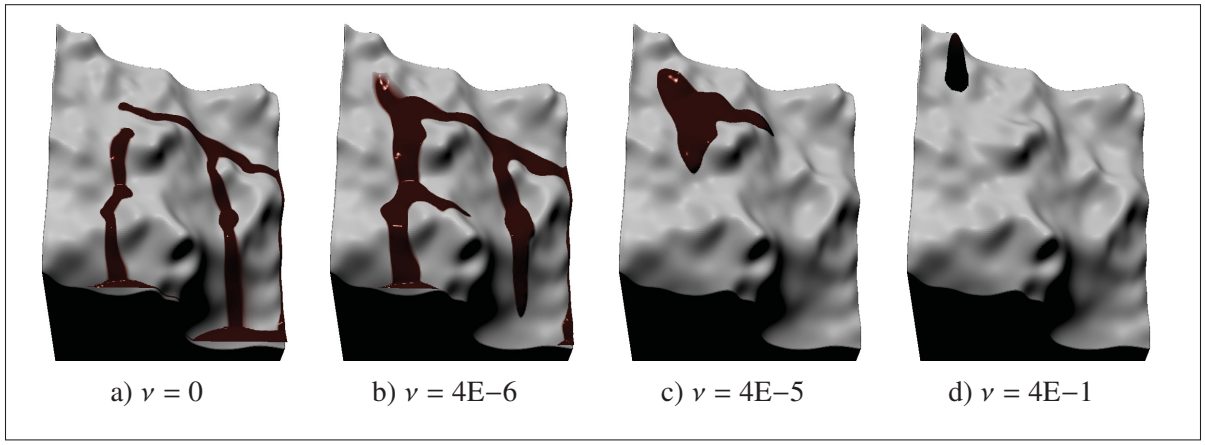


Figure 4.13 A liquid flows on an inclined terrain with varying viscosity, ν, expressed in m^2/s

friction within the liquid. Nevertheless, our model considers the friction at the interface between the liquid and the terrain. While our model does not capture all the features of the complete viscosity model, we can see in our animations that the model produces the behavior expected from a viscous fluid. To further demonstrate that our model captures the important effects of the viscosity, we compared our simulation's behavior with that of a full 3D FLIP simulation (Zhu & Bridson, 2005). Fig. 4.14a-b and the paper's accompanying video show

results for three viscosity values. For matching viscosity values, our approach produces a similar behavior, with a slightly less viscous look. We believe that this difference results from the lack of contribution from neighbor cells. Nonetheless, with a slight adjustment of our viscosity values, our simulation can achieve an even closer match with respect to the FLIP simulation, as shown in Fig 4.14c and in the paper's video.

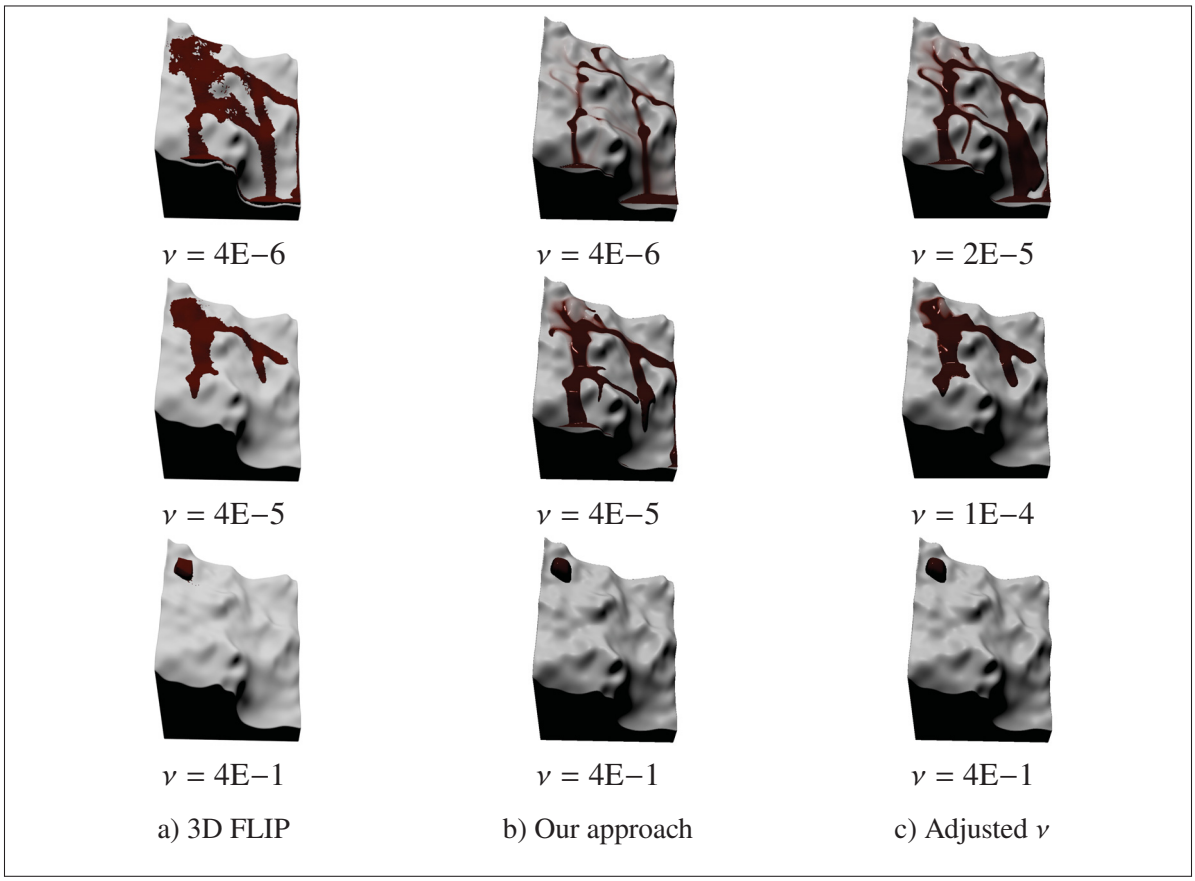


Figure 4.14 Comparison between (a) a 3D FLIP simulation and (b) our approach with the same viscosity values. The behavior is quite similar, and as can be seen in (c), with slight adjustments of the viscosity, our approach can achieve a behavior even closer to that of the FLIP simulation. All images are captured at time $t = 10\ s$. In this scenario, a volume of 1 cm^3 of liquid flows on an inclined surface. The 3D simulation is computed using Houdini's state-of-the-art offline FLIP solver with the same grid spacing as our simulation and an average of eight particles per cell

We make several contributions to the optimization of the liquid surface. As shown in Sec. 4.2, we improve the smoothness of the boundaries, and prevent incorrect interpenetrations. To

demonstrate the surface construction with multiple layers, we show a three-layer scenario with cavities and overhangs in the paper's accompanying video and in Fig. 4.15. Through simulation, the surfaces of the multiple layers link the appropriate columns, even with this configuration exhibiting overhangs and a hole in the middle platform.

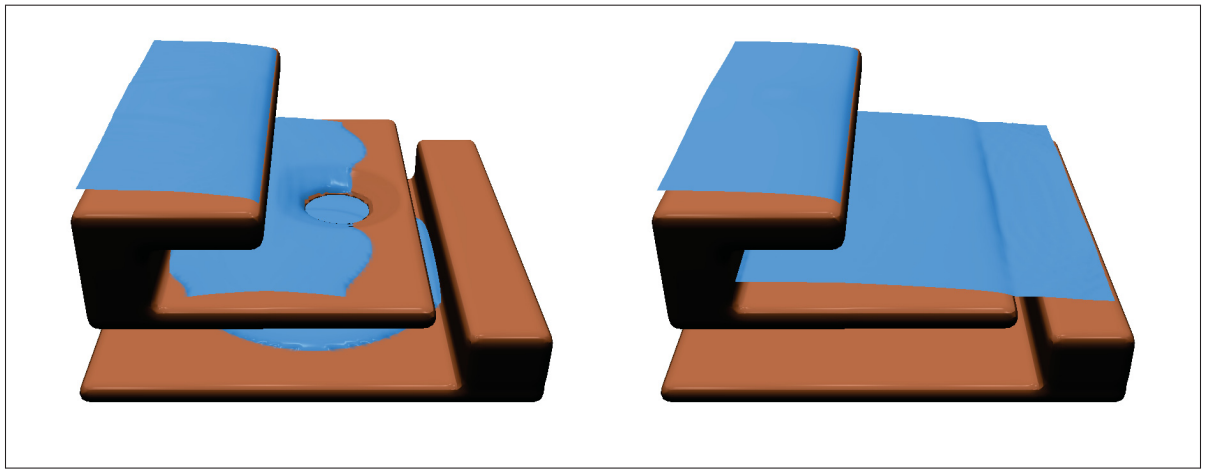
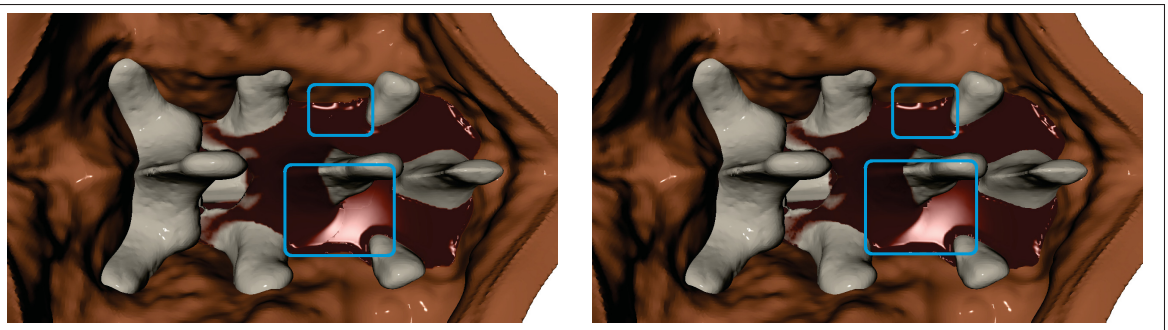


Figure 4.15 This scenario shows how our surface linking approach correctly handles multiple changes in the surface topology

We also validate our approach with a more complex surgery simulation scenario, including multiple overhangs and holes. In Fig. 4.1, blood originates on top of a vertebra during surgery. As the simulation domain is filled, our surface correctly handles the evolving topology of the liquid surface, and no holes or discontinuities are visible.

We compare our surface with that of Borgeat *et al.* (2011) in Fig. 4.16 and in the paper's accompanying video. In their technique, discontinuities in the geometry create seams on the surface near fully submerged overhangs. Furthermore, temporal artifacts can be seen in their video: the fluid surface suffers from popping near the boundary as it spreads. Our fluid surface is free of seams near the overhangs, and our approach preserves a sharp liquid/dry boundary without temporal popping.

We executed the tests presented in this chapter on an Intel Core i5-6600 3.30 GHz with 16 GB of RAM and a GeForce GTX 970. To take advantage of GPU parallelism, our approach was



a) Borgeat *et al.* (2011) b) Our approach

Figure 4.16 Comparison between the surface from Borgeat *et al.* (2011) (left), and our surface (right)

implemented using CUDA. The timings (Table 4.1) show that all of our examples run in real time. Note that the timings also contain the computation for the other contributions from the original paper, which include extended pipes to allows transfer of liquid under fully-flooded passages, and the meniscus rendering on the surface. The most computationally expensive part is the simulation step, which takes around 50–80 % of the total computation time in our examples. As can be seen in Table 4.1 and in Fig. 4.17, the simulation times grow quadratically with respect to the number of cells. All of our examples take considerably less than the 16.6 ms needed to achieve real-time rates, leaving time for other computations such as soft body simulation, haptic feedback, rendering, and collision detection.

4.4 Discussion

In this chapter, we present an approach for real-time fluid simulation in small-scale scenarios. We introduce a novel viscosity model that is both stable and fast to compute. This model is developed using assumptions and approximations that favor stability and speed over accuracy. However, we demonstrate that this novel viscosity model is comparable to state-of-the-art 3D offline models, and that it can be used for any heightmap simulation method, such as the VP or the shallow water equations. Moreover, our approach constructs a triangular mesh surface that accounts for the interlinks between the multiple layers of liquid. Finally, we also introduce a new

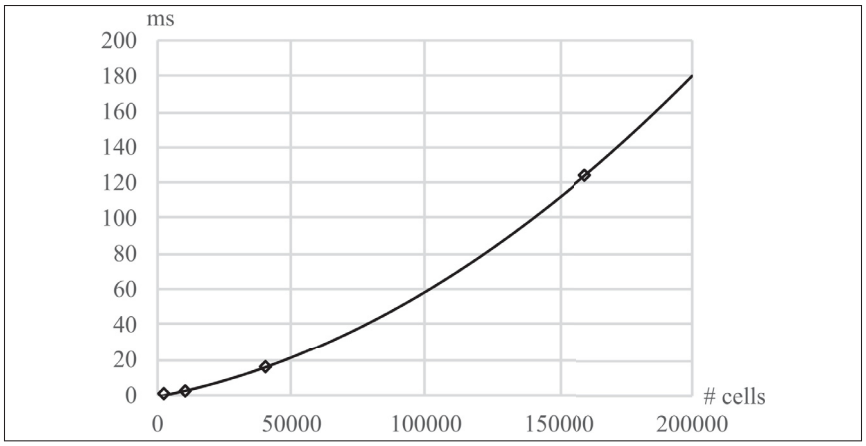


Figure 4.17 The average simulation time per frame (excluding rendering) is quadratic with respect to the number of cells. The figure, as well as its data, have been generated by the co-author of the original paper, Julian Guzman

surface optimization approach that computes minimum heights of the liquid surface in order to prevent interpenetration of the underlying terrain surface. All these contributions improve the realism of small-scale simulations. Furthermore, the approach is computationally inexpensive, which allows it to co-exist with other real-time systems normally required in the context of real applications such as surgery training and games. We have demonstrated that the approach works in real time for various scenarios, such as blood simulation for virtual surgery. This work was part of a collaboration with *OSSimTech*, a company that develops surgery training simulators. The visual quality and plausibility of our results have been validated throughout recurring meetings with their representatives, who provided valuable feedback to guide our research and improve our work.

We show in our paper's accompanying video that our viscosity model and surface reconstruction are independent of the underlying height map simulation; it works just as well with the SWE simulation (Chentanez & Müller, 2010). While the SWE method can take advantage of our contributions, the VP method proved to be a better choice since it had results of equivalent quality with significantly smaller computation times.

The surface constructed by our approach can handle the evolving topology of the liquid throughout the simulation. However, we do not create any geometry to close the gap near edges between unlinked cells (e.g., the edge of an overhang). Nonetheless, these are in areas where the liquid falls down; thus, the water height is generally low, and the gap is not very apparent. Another downside is that generally, the grid does not exactly follow the silhouette at the edge of these overhangs, which sometimes results in having a part of the geometry that will never be covered by the liquid surface. Additionally, in those regions, the liquid falling from one layer to another could be simulated using particles to improve realism, such as in the work of Chentanez & Müller (2010). Finally, the dry/wet boundary moves on a cell-by-cell basis, which can result in some popping. While the depth-based opacity significantly reduces this issue, it is still visible in our examples. This behavior can be improved by increasing the resolution of the grid, or by tracking the surface as suggested by (Müller, Stam, James & Thürey, 2008, Chapter 11).

CONCLUSION AND RECOMMENDATIONS

This thesis focused on preserving the high visual quality of physical simulations of liquid and granular materials, while reducing the computation times and memory requirements. These lessened constraints make it possible to run simulations at a much larger scale or with much more visual details. For offline simulations and rendering, such improvements also enable faster iterations towards the desired artistic vision. As for real-time applications, they leave more resources for the other systems, such as rendering, audio, game mechanics, etc., that coexist with the simulation. To achieve this goal, we explored three distinct approaches, each of which focuses on different aspects of physical simulations for computer graphics.

The approach described in chapter 2 uses an explicit mesh to visualize the surface of a particle-based liquid simulation. The visual features of the initial mesh surface are preserved throughout the simulation until changes in its topology force local remeshing of the surface. In contrast to previous work, our novel detail-preserving projection prevents drifting between the explicit mesh surface while retaining its visual features. This projection is coupled with a novel topology matching stage, which corrects the topology of the explicit mesh surface to match that of an implicit surface built from the particles to ensure that it remains consistent with the underlying particles. Our approach is not limited to a specific particle-based simulation method, and has been tested with both SPH and FLIP simulations, making it well suited for the context of VFX studios who rely heavily on FLIP/APIC simulations. Furthermore, it can be run as a post-process step after the simulation as been computed, which makes it possible to iterate quickly on the final mesh surface generation without having to re-simulate the much more computationally expensive liquid simulation. Our technique is particularly useful for scenarios where the liquid's surface has fine visual details that are expected to be preserved, e.g., melting objects using a highly viscous simulation. Nevertheless, our results showed that our approach is robust enough to handle non-viscous liquids simulations with multiple splashes and topological changes. However, in such cases, it tends to preserve unwanted surface details, e.g., wrinkles

and bumps that would normally be flattened out. Future work could investigate the possibility of detecting such unwanted surface details and smoothing them. Our method could also be improved to better preserve thin sheets of liquids too small to be captured by the volumetric grids used by the topology matching stage. Additionally, our approach sometimes suffers from popping artifacts when changes in topology are detected too late. Future work could also focus on exploring other techniques to detect these changes earlier, thus limiting the size of the mesh surface that needs re-meshing. Finally, the new surface built after topological changes are detected often lacks the finer details that the initial surface had. In that regard, it would be interesting to explore approaches that make use of machine learning to reintroduce finer details.

Chapter 3 introduced an efficient workflow for the simulation of large volumes of snow on the ground that interact with animated characters and objects. In contrast to previous work, the simulation is decoupled into three components: a base layer of snow, snow that interacts with the characters, and finer particles of snow that are projected into the air. This decoupling allows us to use a lightweight representation for the base layer, i.e., untouched snow, which considerably lowers the memory and computation time required to process most of the snow volume. The finer particles are handled using a Semi-Lagrangian fluid simulation, which uses a coarser simulation grid to simulate and visualize very fine snow particles more efficiently. These lightweight representations make it possible to focus most of the memory and computation time for the simulation of snow that interacts with animated characters and objects, i.e., where finer motion and visual details are important. This approach is particularly well suited for the VFX industry. The decoupling adds additional control to fine-tune individual aspects of the simulation. For example, the base layer allows for quick iterations where the volume of snow and the animations can be tweaked to provide the desired imprints. Artists can also control the behavior of the main snow particles independently from the finer particles, and the latter can be iterated several times without requiring an update of the particle-based simulation. While our approach has several advantages, we did identify a few areas that could be improved.

First, our base layer model may generate shapes that are not physically plausible, e.g., large overhangs and large slopes. Future work could be done to detect such cases and convert part of the volume into dynamic snow particles. Moreover, physically-based models could be used to better handle the compression of snow and to better drive the motion of finer snow particles in the Semi-Lagrangian fluid simulation. Finally, we believe that memory requirements could be further reduced by detecting static snow particles and transferring their volume back into the base layer. However, care must be taken to avoid flickering, and it would also remove the ability to run the full base layer simulation prior to running the particle-based simulation.

In chapter 4, we proposed a novel approach for simulating small-scale liquids in real-time. It was developed as part of a project that aimed to simulate blood within a surgeon training software, which often involves thin sheets of liquid that flow on a surface. To achieve this goal, our approach improves upon previous work that use a 2.5D simulation of columns of liquids by introducing an enhanced multi-layered surface reconstruction that better handles overhangs and changes in topology. The 2.5D heightmap simulation can capture arbitrarily thin sheets of liquids with much fewer simulation elements, while our multi-layered surface can cover a much wider range of scenarios that can be seen in the real world. We also introduce a new surface optimization process that prevents unwanted interpenetrations between the liquid surface and the underlying geometry. Furthermore, we presented a novel physically-based viscosity model that is both stable and fast to compute. Because viscosity has a bigger impact on the blood's motion for small-scale scenarios, this model improves the realism of the simulation. Although this formulation relies on several assumptions that lower its accuracy, our results show that it produces results that are visually similar to that of an offline 3D simulation, at a fraction of the computation cost and memory. However, to match the offline simulation, the viscosity value had to be manually adjusted. In the future, it would be interesting to derive a model that computes the adjusted viscosity value that would match the ground truth. Moreover, the surface reconstruction

could be improved to better follow the silhouette at the edge of overhangs, and to limit visual popping artifacts that can sometimes be seen as the dry/wet boundary moves between grid cells.

The approaches described in this thesis enhance the visual quality of physical simulations using different strategies. In chapter 2, the visualization is decoupled from the simulation, allowing us to use a representation that better captures the finer details in the visualization. In chapter 3, the simulation is decomposed into separate paradigms to allow using the most efficient techniques, and also to focus resources where it matters the most. Finally, in chapter 4 a 2.5D simulation is used for a 3D liquid, along with a simplified physical model to handle viscosity in order to run in real-time. Although these strategies have been applied on very specific problems, their core ideas can also be applied to a variety of applications that involve physical simulations for computer graphics.

LIST OF REFERENCES

Aanjaneya, M., Gao, M., Liu, H., Batty, C. & Sifakis, E. (2017). Power Diagrams and Sparse Paged Grids for High Resolution Adaptive Liquids. *ACM Trans. Graph.*, 36(4), 140:1–140:12.

Adams, B., Pauly, M., Keiser, R. & Guibas, L. J. (2007). Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3), 48–es.

Alduán, I. & Otaduy, M. A. (2011). SPH granular flow with friction and cohesion. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 25–32.

Ando, R. & Batty, C. (2020). A practical octree liquid simulator with adaptive surface resolution. *ACM Trans. Graph.*, 39(4).

Ando, R., Thürey, N. & Wojtan, C. (2013). Highly Adaptive Liquid Simulations on Tetrahedral Meshes. *ACM Trans. Graph.*, 32(4), 103:1–103:10.

Andreasson, S., Östergaard, L. & Goswami, P. (2024). Parallel spatiotemporally adaptive DEM-based snow simulation. *Proc. ACM Comput. Graph. Interact. Tech.*, 7(3).

Angst, R., Thuerey, N., Botsch, M. & Gross, M. (2008). Robust and Efficient Wave Simulations on Deforming Meshes. *Computer Graphics Forum*, 27(7), 1895–1900.

Becker, M. & Teschner, M. (2007). Weakly compressible SPH for free surface flows. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '07), 209–217.

Bell, N., Yu, Y. & Mucha, P. J. (2005). Particle-based Simulation of Granular Materials. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 77–86.

Bender, J. & Koschier, D. (2017). Divergence-Free SPH for Incompressible and Viscous Fluids. *IEEE Trans. on Visualization and Computer Graphics*, 23(3), 1193-1206.

Bhatacharya, H., Gao, Y. & Bargteil, A. (2011). A Level-set Method for Skinning Animated Particle Data. *Proc. of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '11), 17–24.

Borgeat, L., Massicotte, P., Poirier, G. & Godin, G. (2011). Layered Surface Fluid Simulation for Surgical Training. *Proc. of Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011 Conference*, pp. 323–330.

Boukadida, N., Andrews, S. & Paquette, E. (2019). Poster: Interactive soft-body simulation for surgical applications. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

Brochu, T. & Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4), 2472–2493.

Bronstein, A., Bronstein, M. & Kimmel, R. (2008). *Numerical Geometry of Non-Rigid Shapes*. Springer.

Chentanez, N. & Müller, M. (2010). Real-time Simulation of Large Bodies of Water with Small Scale Details. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '10), 197–206.

Chentanez, N., Müller, M., Macklin, M. & Kim, T.-Y. (2015). Fast grid-free surface tracking. *ACM Trans. Graph.*, 34(4).

Da, F., Batty, C. & Grinspun, E. (2014). Multimaterial Mesh-based Surface Tracking. *ACM Trans. Graph.*, 33(4), 112:1–112:11.

Dagenais, F., Gagnon, J. & Paquette, E. (2012). A Prediction-Correction Approach for Stable SPH Fluid Simulation from Liquid to Rigid. *Proceedings of Computer Graphics International 2012*.

Dagenais, F., Gagnon, J. & Paquette, E. (2016). An Efficient Layered Simulation Workflow For Snow Imprints. *The Visual Computer*, 32, accepted for publication.

Dagenais, F., Gagnon, J. & Paquette, E. (2017). Detail-Preserving Explicit Mesh Projection and Topology Matching for Particle-Based Fluids. *Computer Graphics Forum*, 36, 444–457.

Dagenais, F., Guzmán, J., Vervondel, V., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018a). Extended virtual pipes for the stable and real-time simulation of small-scale shallow water. *Computers & Graphics*, 76, 84–95.

Dagenais, F., Guzmán, J., Vervondel, V., Hay, A., Delorme, S., Mould, D. & Paquette, E. (2018b). Real-Time Virtual Pipes Simulation and Modeling for Small-Scale Shallow Water. *Proceeginds of VRIPHYS 2018 Workshop on Virtual Reality Interaction and Physical Simulation*.

Desbrun, M., Cani, M.-P. et al. (1996). Smoothed particles: A new paradigm for animating highly deformable bodies. *Proc. of the Eurographics workshop on Computer animation and simulation*, 96, 61–76.

Edwards, E. & Bridson, R. (2014). Detailed Water with Coarse Grids: Combining Surface Meshes and Adaptive Discontinuous Galerkin. *ACM Trans. Graph.*, 33(4), 136:1–136:9.

Enright, D., Marschner, S. & Fedkiw, R. (2002). Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.*, 21(3), 736–744.

Fearing, P. (2000). Computer modelling of fallen snow. *Proceedings of SIGGRAPH 00*, (Annual Conference Series), 37–46.

Feldman, B. E. & O'Brien, J. F. (2002). Modeling the accumulation of wind-driven snow. *SIGGRAPH 2002 Conference Abstracts and Applications*, pp. 218–218.

Ferstl, F., Ando, R., Wojtan, C., Westermann, R. & Thuerey, N. (2016). Narrow Band FLIP for Liquid Simulations. *Computer Graphics Forum*, 35(2), 225–232.

Festenberg, N. v. & Gumhold, S. (2009). A geometric algorithm for snow distribution in virtual scenes. *Eurographics Workshop on Natural Phenomena*, pp. 15–25.

Festenberg, N. v. & Gumhold, S. (2011). Diffusion-Based Snow Cover Generation. *Computer Graphics Forum*, 30(6), 1837–1849.

Foster, N. & Fedkiw, R. (2001). Practical Animation of Liquids. *Proceedings of SIGGRAPH 01*, pp. 23–30.

Foster, N. & Metaxas, D. (1996). Realistic Animation of Liquids. *Graphical Models and Image Processing*, 58(5), 471–483.

Gagnon, J., Dagenais, F. & Paquette, E. (2016). Dynamic Lapped Texture for Fluid Simulations. *The Visual Computer*, 32, accepted for publication.

Gagnon, J., Guzmán, J., Vervondel, V., Dagenais, F., Mould, D. & Paquette, E. (2019). Distribution Update of Deformable Patches for Texture Synthesis on the Free Surface of Fluids. *Computer Graphics Forum*, 38, 491–500.

Gagnon, J., Guzmán, J., Mould, D. & Paquette, E. (2021). Patch Erosion for Deformable Lapped Textures on 3D Fluids. *Computer Graphics Forum*, 40, 367-374.

Gissler, C., Henne, A., Band, S., Peer, A. & Teschner, M. (2020). An Implicit Compressible SPH Solver for Snow Simulation. *ACM Trans. Graph.*, 39(4).

Goldade, R., Batty, C. & Wojtan, C. (2016). A Practical Method for High-Resolution Embedded Liquid Surfaces. *Computer Graphics Forum*, 35(2), 233–242.

Han, X., Xu, C. & Prince, J. L. (2003). A topology preserving level set method for geometric deformable models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(6), 755-768.

Hart, J. C. (1998). Morse Theory for Implicit Surface Modeling. In *Mathematical Visualization: Algorithms, Applications and Numerics* (pp. 257–268). Springer Berlin Heidelberg.

Heiss-Synak, P., Kalinov, A., Strugaru, M., Etemadi, A., Yang, H. & Wojtan, C. (2024). Multi-Material Mesh-Based Surface Tracking with Implicit Topology Changes. *ACM Trans. Graph.*, 43(4).

Ihmsen, M., Wahl, A. & Teschner, M. (2013). A Lagrangian framework for simulating granular material with high detail. *Computers & Graphics*, 37(7), 800–808.

Ihmsen, M., Cornelis, J., Solenthaler, B., Horvath, C. & Teschner, M. (2014a). Implicit incompressible SPH. *IEEE Trans. on Visualization and Computer Graphics*, 20(3), 426–35.

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A. & Teschner, M. (2014b). SPH Fluids in Computer Graphics. *Eurographics 2014 - State of the Art Reports*.

Jeschke, S. & Wojtan, C. (2017). Water Wave Packets. *ACM Trans. Graph.*, 36(4), 103:1–103:12.

Jiang, C., Schroeder, C., Selle, A., Teran, J. & Stomakhin, A. (2015). The Affine Particle-in-cell Method. *ACM Trans. Graph.*, 34(4), 51:1–51:10.

Kass, M. & Miller, G. (1990). Rapid, Stable Fluid Dynamics for Computer Graphics. *Comput. Graph.*, 24(4), 49–57.

Kellomäki, T. (2014). Rigid Body Interaction for Large-Scale Real-Time Water Simulation. *International Journal of Computer Games Technology*, 2014(1).

Kim, D., Song, O.-y. & Ko, H.-S. (2009). Stretching and Wiggling Liquids. *ACM Trans. Graph.*, 28(5), 120:1–120:7.

Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C. & Teran, J. (2016). Drucker-prager elastoplasticity for sand animation. *ACM Trans. Graph.*, 35(4).

Klár, G., Budsberg, J., Titus, M., Jones, S. & Museth, K. (2017). Production ready MPM simulations. *ACM SIGGRAPH 2017 Talks*, (SIGGRAPH '17).

Layton, A. T. & van de Panne, M. (2002). A Numerically Efficient and Stable Algorithm for Animating Water Waves. *The Visual Computer*, 18(1), 41–53.

Lee, R. & O'Sullivan, C. (2007). A Fast and Compact Solver for the Shallow Water Equations. *Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS)*, pp. 51–57.

Lenaerts, T. & Dutré, P. (2009). Mixing fluids and granular materials. *Computer Graphics Forum*, 28(2), 213–218.

Lorensen, W. E. & Cline, H. E. (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, 21(4), 163–169.

Losasso, F., Gibou, F. & Fedkiw, R. (2004). Simulating Water and Smoke with an Octree Data Structure. *ACM Trans. Graph.*, 23(3), 457–462.

Macklin, M. & Müller, M. (2013). Position Based Fluids. *ACM Trans. Graph.*, 32(4), 104:1–104:12.

Macklin, M., Müller, M., Chentanez, N. & Kim, T.-Y. (2014). Unified Particle Physics for Real-time Applications. *ACM Trans. Graph.*, 33(4), 153:1–153:12.

Maréchal, N., Guérin, E., Galin, E., Mérillou, S. & Mérillou, N. (2010). Heat transfer simulation for modeling realistic winter sceneries. *Computer Graphics Forum*, 29(2), 449–458.

Mei, X., Decaudin, P. & Hu, B.-G. (2007, Oct). Fast Hydraulic Erosion Simulation and Visualization on GPU. *Pacific Conference on Computer Graphics and Applications, PG '07*, pp. 47-56.

Moeslund, T. B., Madsen, C. B., Aagaard, M. & Lerche, D. (2005). Modeling falling and accumulating snow. *Vision, Video and Graphics*, 2.

Montani, C., Scateni, R. & Scopigno, R. (1994). A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6), 353–355.

Mould, D. & Yang, Y.-H. (1997). Modeling Water for Computer Graphics. *Computers & Graphics*, 21(6), 801–814.

Müller, M. (2009). Fast and robust tracking of fluid surfaces. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '09), 237–245.

Müller, M., Charypar, D. & Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '03), 154–159.

Müller, M., Heidelberger, B., Hennix, M. & Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2), 109–118.

Müller, M., Stam, J., James, D. & Thürey, N. (2008). Real Time Physics. *SIGGRAPH Class Notes*, pp. 88:1–88:90.

Museth, K., Breen, D. E., Whitaker, R. T. & Barr, A. H. (2002). Level Set Surface Editing Operators. *ACM Trans. Graph.*, 21(3), 330–338.

Museth, K., Lait, J., Johanson, J., Budsberg, J., Henderson, R., Alden, M., Cucka, P., Hill, D. & Pearce, A. (2013). OpenVDB: an open-source data structure and toolkit for high-resolution volumes. *SIGGRAPH 2013 Courses*, pp. 19.

Narain, R., Golas, A. & Lin, M. C. (2010). Free-flowing Granular Materials with Two-way Solid Coupling. *ACM Trans. Graph.*, 29(6), 173:1–173:10.

O'Brien, J. F. & Hodgins, J. K. (1995, Apr). Dynamic Simulation of Splashing Fluids. *Proc. of Computer Animation '95*, pp. 198-205, 220.

Onoue, K. & Nishita, T. (2005). An interactive deformation system for granular material. *Computer Graphics Forum*, 24(1), 51–60.

Pla-Castells, M., García-Fernández, I., Martinez-Dura, R. et al. (2008). Physically-based interactive sand simulation. *Eurographics 2008, Short Papers*, pp. 21–24.

Qu, Z., Li, M., De Goes, F. & Jiang, C. (2022). The power particle-in-cell method. *ACM Trans. Graph.*, 41(4).

Roy, B., Paquette, E. & Poulin, P. (2020). Particle upsampling as a flexible post-processing approach to increase details in animations of splashing liquids. *Computers & Graphics*, 88, 57-69.

Roy, B., Poulin, P. & Paquette, E. (2021). Neural UpFlow: A Scene Flow Learning Approach to Increase the Apparent Resolution of Particle-Based Liquids. *Proc. ACM Comput. Graph. Interact. Tech.*, 4(3).

Sato, T., Wojtan, C., Thuerey, N., Igarashi, T. & Ando, R. (2018). Extended Narrow Band FLIP for Liquid Simulations. *Computer Graphics Forum*, 37(2), 169-177.

Sellán, S., Aigerman, N. & Jacobson, A. (2021). Swept volumes via spacetime numerical continuation. *ACM Trans. Graph.*, 40(4).

Selle, A., Fedkiw, R., Kim, B., Liu, Y. & Rossignac, J. (2008). An unconditionally stable MacCormack method. *Journal of Scientific Computing*, 35(2-3), 350–371.

Sethian, J. A. (1995). A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proc. Nat. Acad. Sci*, pp. 1591–1595.

Sivakumaran, G., Paquette, E. & Mould, D. (2022). Time Reversal and Simulation Merging for Target-Driven Fluid Animation. *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games*, (MIG '22).

Solenthaler, B., Schläfli, J. & Pajarola, R. (2007). A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*, 18(1), 69–82.

Stam, J. (1999). Stable Fluids. *Proceedings of SIGGRAPH 99*, (Annual Conference Series), 121–128.

Stander, B. T. & Hart, J. C. (1997). Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling. *Proceedings of SIGGRAPH 97*, (Annual Conference Series), 279–286.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J. & Selle, A. (2013). A material point method for snow simulation. *ACM Trans. Graph.*, 32(4), 102.

Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J. & Selle, A. (2014). Augmented MPM for phase-change and varied materials. *ACM Trans. Graph.*, 33(4).

Su, H., Xue, T., Han, C., Jiang, C. & Aanjaneya, M. (2021). A unified second-order accurate in time MPM formulation for simulating viscoelastic liquids with phase change. *ACM Trans. Graph.*, 40(4).

Su, H., Zhang, S., Pan, Z., Aanjaneya, M., Gao, X. & Wu, K. (2023). Real-time Height-field Simulation of Sand and Water Mixtures. *SIGGRAPH Asia 2023 Conference Papers*, (SA '23).

Sumner, R. W., O'Brien, J. F. & Hodgins, J. K. (1999). Animating sand, mud, and snow. *Computer Graphics Forum*, 18(1), 17–26.

Takahashi, T. & Fujishiro, I. (2012). Particle-based simulation of snow trampling taking sintering effect into account. *SIGGRAPH 2012 Posters*, pp. 7.

Šťava, O., Beneš, B., Brisbin, M. & Křivánek, J. (2008). Interactive Terrain Modeling Using Hydraulic Erosion. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '08), 201–210.

Wang, C., Wang, Z., Xia, T. & Peng, Q. (2006). Real-time snowing simulation. *The Visual Computer*, 22(5), 315–323.

Wang, H., Miller, G. & Turk, G. (2007). Solving General Shallow Wave Equations on Surfaces. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (SCA '07), 229–238.

Williams, B. (2008). *Fluid surface reconstruction from particles*. (Master's thesis, The University of British Columbia).

Winchenbach, R. & Kolb, A. (2021). Optimized Refinement for Spatially Adaptive SPH. *ACM Trans. Graph.*, 40(1).

Winchenbach, R., Hochstetter, H. & Kolb, A. (2017). Infinite continuous adaptivity for incompressible SPH. *ACM Trans. Graph.*, 36(4).

Wojtan, C., Thürey, N., Gross, M. & Turk, G. (2009). Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3), 76:1–76:10.

Wojtan, C., Thürey, N., Gross, M. & Turk, G. (2010). Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph.*, 29(4), 50:1–50:8.

Wojtan, C., Müller-Fischer, M. & Brochu, T. (2011). Liquid Simulation with Mesh-based Surface Tracking. *ACM SIGGRAPH 2011 Courses*, pp. 8:1–8:84.

Wong, S.-K., Fu, I. et al. (2015). Hybrid-based snow simulation and snow rendering with shell textures. *Computer Animation and Virtual Worlds*, 26(3-4), 413–421.

Yang, S. & Gobbert, M. K. (2009). The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions. *Applied Mathematics Letters*, 22(3), 325–331.

Yu, C., Li, X., Lan, L., Yang, Y. & Jiang, C. (2024). XPBI: Position-Based Dynamics with Smoothing Kernels Handles Continuum Inelasticity. *SIGGRAPH Asia 2024 Conference Papers*, (SA '24).

Yu, J. & Turk, G. (2013). Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph.*, 32(1), 5:1–5:12.

Yu, J., Wojtan, C., Turk, G. & Yap, C. (2012). Explicit Mesh Surfaces for Particle Based Fluids. *Computer Graphics Forum*, 31, 815–824.

Yuksel, C., House, D. H. & Keyser, J. (2007). Wave Particles. *ACM Trans. Graph.*, 26(3).

Zeng, Y.-L., Tan, C. I., Tai, W.-K., Yang, M.-T., Chiang, C.-C. & Chang, C.-C. (2007). A momentum-based deformation system for granular material. *Computer Animation and Virtual Worlds*, 18(4-5), 289–300.

Zhu, B. & Yang, X. (2010). Animating sand as a surface flow. *Eurographics 2010, Short Papers*.

Zhu, K., He, X., Li, S., Wang, H. & Wang, G. (2021). Shallow Sand Equations: Real-Time Height Field Simulation of Dry Granular Flows. *IEEE Transactions on Visualization and Computer Graphics*, 27(3), 2073-2084.

Zhu, Y. & Bridson, R. (2005). Animating sand as a fluid. *ACM Trans. Graph.*, 24(3), 965–972.

Zorin, D., Schröder, P. & Sweldens, W. (1996). Interpolating Subdivision for Meshes with Arbitrary Topology. *Proceedings of SIGGRAPH 96*, pp. 189–192.