Enhancing Fluid Animation with Fine Detail

Von der Fakultät für Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Dieter Hubertus Valentin Morgenroth

aus Scheßlitz

Hauptberichter:	Prof. Dr. Daniel Weiskopf
Mitberichter:	Prof. Dr. Bernhard Eberhardt

Tag der mündlichen Prüfung: 16.12.2021

Visualisierungsinstitut
der Universität Stuttgart

2021

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

Abbreviations

2D	two-dimensional
3D	three-dimensional
AI	artificial intelligence
AWS	Amazon Web Services
BVH	bounding volume hierarchy
CFL	Courant–Friedrichs–Lewy
CG	computer generated
CGI	computer generated images
CPU	central processing unit
CPM	closest point method
GPU	graphics processing unit
IoR	index of refraction
IK	inverse kinematics
ML	machine learning
ODE	ordinary differential equation
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing

PCISPH	predictive-corrective incompressible Smoothed Particle Hydrodynamics
PDE	partial differential equation
RPC	remote procedure call
SDK	software development kit
SOP	surface operator
SPH	Smoothed Particle Hydrodynamics
VFX	visual effects

Units

MB	Megabyte

List of Symbols

Operators

Symbol	Explanation
$\nabla\cdot$	divergence: $\nabla\cdot = \left(\frac{\partial}{\partial x_1} + ... + \frac{\partial}{\partial x_n}\right)$
∇	gradient: $\nabla = \left(\frac{\partial}{\partial x_1}, ..., \frac{\partial}{\partial x_n}\right)^T$
∇^2	Laplacian: $\nabla^2 = \nabla \cdot \nabla = \left(\frac{\partial^2}{\partial x_1^2} + ... + \frac{\partial^2}{\partial x_n^2}\right)$
$\frac{D}{Dt}$	material derivative: $\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla$
$\|\|.\|\|$	Euclidean norm
$\dot{\mathbf{x}}$	first time derivative $\dot{\mathbf{x}} = \frac{\partial \mathbf{x}}{\partial t}$
$\otimes$	tensor product of two vector spaces

Greek Symbols

Symbol	Units	Explanation
α_Y	deg	contact angle
β	deg	tilt angle of the wall
η	-	coefficient for thermal expansion
γ	$\mathrm{N\,m^{-1}}$	interface tension
κ	$\mathrm{m^2\,s^{-2}}$	(pressure) stiffness constant
μ	$\mathrm{kg\,m^{-1}\,s^{-1}}$	dynamic viscosity
μ_d	$\mathrm{m^2\,s^{-1}}$	diffusion coefficient
ν	$\mathrm{m^2\,s^{-1}}$	kinematic viscosity
ψ	deg	angle to horizontal plane
ρ	$\mathrm{kg\,m^{-3}}$	mass density
σ	$\mathrm{N\,m^{-1}}$	interfacial tension

Symbol	Units	Explanation
σ	$\mathrm{kg\,m^{-3}\,s^{-1}}$	substance generation coefficient
$\mathcal{T}$	K	temperature

Latin Symbols a-z, A-Z

Convention throughout this work is to write scalar values with normal characters and vector values with **bold** characters.

Symbol	Units	Explanation
$\mathbf{a}$	$\mathrm{m\,s^{-2}}$	acceleration
dv	-	volume integration variable
d	m	correction length
$\mathbf{f}$	N	external force acting on fluid
$\mathbf{g}$	$\mathrm{m\,s^{-2}}$	gravitational acceleration
h	m	kernel smoothing length
l	m	capillary length
m	kg	mass
$\mathbf{n}$	m	normal on surface
p	$\mathrm{N\,m^{-2}}$	pressure
$\mathbf{p}$	m	point on surface
r	m	distance to particle
t	s	time
$\mathbf{u}$	$\mathrm{m\,s^{-1}}$	velocity with $\mathbf{u} = (u,v,w)^T$
$\mathbf{v}$	*	arbitrary vector value
$\mathbf{x}$	m	position in world coordinates
y	m	distance above horizontal level
H	$\mathrm{m^{-1}}$	mean curvature
M	-	evolving surface space
O	-	map that relates elements between time steps
$T_{\mathbf{p}}M$	$\mathrm{m^{-3}}$	tangent space at point $\mathbf{p}$
$W(r,h)$	-	kernel function with distance r and support radius h

Acknowledgments

To begin with, I would like to express my gratitude to Daniel Weiskopf and Bernd Eberhardt for letting me start this adventure despite my advancing age, long break from academics, and limited time budget. I am grateful for your support throughout this process. Thank you for allowing me to be your Ph.D. student as an external part time student. I always felt welcome and enjoyed every Tuesday when I was joining the weekly meetings. Thank you for your patience on this long journey. Your guidance and example improved both my academic work and my personality.

I want to thank the Kooperatives Promotionskolleg Digital Media for accepting me as an external student. I enjoyed being part of it, and I profited a lot from the excellent atmosphere of the team: Jan Fröhlich, Lena Gieseke, Sebastian Herholz, Markus Huber, David Koerner, Tim Krake, Stefan Reinhardt, and Benjamin Wollet.

I also want to thank my co-authors and collaborators who helped with my work: Stefan Reinhardt and Mariusz Wesirski.

I am thankful for the hospitality I received from the colleagues at the HdM Media University: Jochen Bomm, Johannes Schaugg, Andreas Schmid, Ingmar Rieger, and Robin Schulte.

Thank you, Markus Huber, for providing the LATEX template for this thesis.

I am deeply thankful for all the family members and friends who supported me during this time.

–Dieter Morgenroth

Abstract

Water, or liquids in general, are popular ingredients for action scenes in movies. Therefore, in the field of computer graphics for visual effects (VFX), the simulation and rendering of liquids is a frequently required task. The liquid simulations found in movies are mostly large-scale and cover length scales in the meter to kilometer range. However, there are effects of the physics of liquids that take place on a small scale, but that can also have a noticeable optical effect in large-scale scenes. The calculation of these small-scale effects requires a very high resolution and is therefore often not possible when simulating large scenes for reasons of time and costs. This dissertation discusses strategies to enhance large-scale fluid simulations with small-scale physical effects.

The first part of this dissertation describes a VFX production pipeline for liquid simulations. The new possibilities of cloud computing are used to increase the resolution thanks to expanded computing power and thus to achieve more details. Using a client/server architecture and remote procedure calls, a system was set up that enables interactive work on a simulation scene in a local application in which the complex calculations of the simulation take place on an outsourced computer in the cloud. Another contribution within the framework of this system is the introduction of "blind particles" that make it possible to delete unnecessary particles from data sets without affecting the visual result. This can save bandwidth and rendering time.

The second part of the dissertation presents a direct ray tracing method for implicitly described liquid surfaces that takes into account the capillary effects at the interfaces to solids. The method uses the analytical solution of the meniscus shape at the fluid interface to achieve the effect of surface tension between the water surface and the solid. It generates correct contact angles at the edges without the need for a computationally intensive simulation. At render time, it combines the analytical solution for a small-scale effect with the numerical solution for a large-scale simulation. The process guarantees the correct contact angle and, in certain scenarios, delivers the correct solution

across the entire interface; even in general scenarios, it delivers plausible results.

In the last part, a method is presented to simulate fluid flows on developing surfaces, e.g., an oil film on a water surface. In the case of an animated surface (e.g., extracted from a particle-based fluid simulation) in three-dimensional space, a second simulation is added to the input surface. In general, a partial differential equation is solved on a level set surface. Coupling strategies between input properties and simulation are introduced, and the conservation of mass and momentum is added to existing methods. In this way, high-resolution two-dimensional simulations on coarse input surfaces are efficiently calculated.

German Abstract
—Zusammenfassung—

Wasser oder allgemein Flüssigkeiten sind beliebte Bestandteile für Actionszenen in Filmen. Deshalb ist im Bereich der Computergrafik für visuelle Effekte (VFX) die Simulation und das Rendern von Flüssigkeiten eine häufig benötigte Fähigkeit. Die Flüssigkeitssimulationen sind dabei meist groß angelegt und decken Längenskalen im Meter- bis Kilometerbereich ab. Es gibt aber physikalische Effekte von Flüssigkeiten, die sich im kleinen Maßstab abspielen, aber eine auffällige optische Wirkung auch im Großen haben können. Die Berechnung dieser kleinskaligen Effekte benötigt eine sehr hohe Auflösung und ist deshalb bei der Simulation großer Szenen aus Zeit- und Kostengründen oft nicht möglich. Diese Dissertation diskutiert Strategien, um Fluidsimulationen mit kleinskaligen physikalischen Effekten zu ergänzen.

Der erste Teil dieser Dissertation beschreibt eine VFX-Produktionspipeline für Flüssigkeitssimulationen, die die neuen Möglichkeiten von Cloud-Computing-Angeboten ausnutzt, um die Auflösung dank erweiterter Rechenleistung zu erhöhen und so mehr Details zu erreichen. Dabei wurde mittels einer Client/Serverarchitektur und Remote-Procedure-Calls ein System aufgebaut, das interaktives Arbeiten an einer Simulationsszene in einer lokalen Applikation ermöglicht, in dem die aufwendigen Berechnungen der Simulation auf einem ausgelagerten Rechner in der Cloud stattfinden. Dabei wurde auch auf spezielle GPU-Hardware zurückgegriffen. Ein weiterer Beitrag im Rahmen dieses Systems ist die Einführung von “Blind Particles”, bei deren Verwendung es möglich wird, unnötige Partikel aus Datensätzen zu löschen, ohne das visuelle Ergebnis zu beeinflussen. Dadurch kann Bandbreite und Renderzeit gespart werden.

Der zweite Teil der Dissertation stellt eine direkte Raytracing-Methode für implizit beschriebene Flüssigkeitsoberflächen vor, die die Kappilareffekte an den Grenzflächen zu Festkörpern berücksichtigt. Das Verfahren verwendet die analytische Lösung der Meniskusform an der Fluidgrenzfläche, um den Effekt der Oberflächenspannung zwischen Wasseroberfläche und Festkörper zu erzielen. Das Verfahren erzeugt korrekte Kontaktwinkel an den Rändern,

ohne dass eine rechenintensive Simulation erforderlich ist. Zur Renderzeit kombiniert es die analytische Lösung für einen kleinskaligen Effekt mit der numerischen Lösung einer großskaligen Simulation. Das Verfahren garantiert den richtigen Kontaktwinkel und liefert in bestimmten Szenarien die richtige Lösung über die gesamte Grenzfläche; selbst in allgemeinen Szenarien liefert es plausible Ergebnisse.

Im letzten Teil wird ein Verfahren vorgestellt, um Fluidströmungen auf sich entwickelnden Oberflächen zu simulieren, z.B. einen Ölfilm auf einer Wasseroberfläche. Bei einer animierten Oberfläche (z.B. extrahiert aus einer partikelbasierten Fluidsimulation) im dreidimensionalen Raum wird eine zweite Simulation auf der Eingabeoberfläche hinzugefügt. Im Allgemeinen wird eine partielle Differentialgleichung auf einer Level-Set-Oberfläche gelöst. Es werden Kopplungsstrategien zwischen Eingabeeigenschaften und der Simulation eingeführt, und Masse- und Impulserhaltung wird zu bestehenden Methoden hinzugefügt. Auf diese Weise werden hochauflösende 2D-Simulationen auf groben Eingabeflächen effizient berechnet.

CHAPTER 1

Introduction

In the last three decades, fluid simulation for visual effects has seen significant advances in the domain of free surface motion. Movies like Waterworld (1995) and Titanic (1997) were the first prominent movies that used computer generated (CG) water effects to show large-scale ocean surfaces. In those two movies, the CG water was relatively calm. The actual sinking ship was still shot with miniatures. However, the technology advanced, and the fluid simulations became more and more convincing. Already nine years after the Titanic movie, for the movie 300, sinking ships were fully simulated.

However, in the VFX domain, most researchers worked in the area of large- to mid-scale simulations where the small-scale effects on the surface and at the interface to solid boundaries are not dominant features. When computing large-scale simulations, the forces of surface tension are negligible. Putting computational work into their calculation is therefore not reasonable.

To simulate small-scale effects on the fluid surface, e.g., a soap film on a water surface on large-scale simulations seems to be computational overkill. However, it is these small details that can make all the difference in making a scene appear photorealistic. One example of such a small detail can be seen when looking at a glass of orange juice as photographed in Fig. 1.1. Notice the little highlight on the edge where the fluid surface is touching the glass surface. This highlight is created by the fact that the surface is slightly curved due to the meniscus effect and reflects light sources from a wide angle. Another small detail that could add a lot of realism to a scene is the rainbow colors caused by an oil spill on the surface of the water.

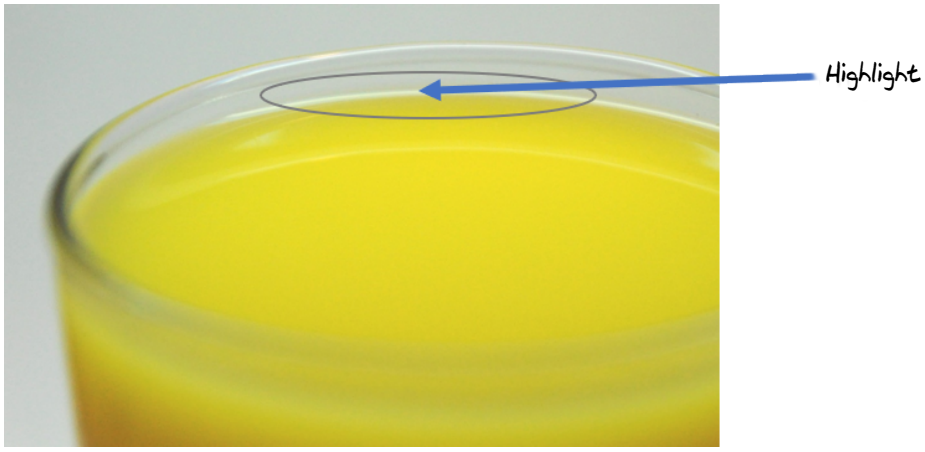


Figure 1.1: The small-scale meniscus effect creates a highlight on the water surface and is a dominant feature in mid-scale fluid scenes.

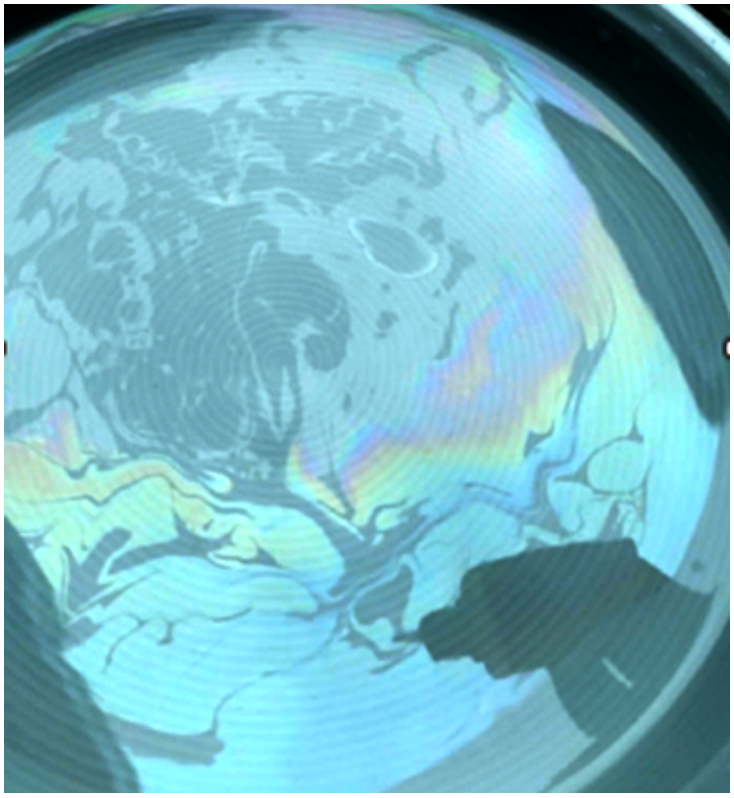


Figure 1.2: The light interference thin film effect is caused by differences in the oil film thickness in the micrometer scale. However, the rainbow effect this causes is well visible in the mid-scale scene.

As shown in the glass of orange juice in Fig. 1.1 and the oil spill in Fig. 1.2, these small effects can have a big impact on the image and the achieved level of photorealism. The glass of orange juice would look wrong if the highlight at the curved meniscus was missing. In Fig. 1.2, the water surface with the shimmering rainbow colors creates a completely different image of the surface, although the oil film is only a few micrometers thick, which is negligible compared to the overall fluid body.

1.1 RESEARCH CHALLENGES

When using fluid simulation for visual effects, the main goal is to create convincing animations within budget. It does not matter whether the animation is physically correct. Adding small-scale effects to fluid simulation could add additional realism to the images. When there are small-scale physical effects that are worth adding to achieve greater realism, first, an adequate physical model has to be found and then an efficient numerical method has to be chosen. This work explores different approaches to add small-scale effects in a computer generated images (CGI) production in a cost-effective way.

Overall Goal

What are effective strategies to augment fluid animations with small-scale physical effects?

The traditional approach for generating small-scale effects is to simply increase the resolution of the simulation so it can capture the small details better. No matter which numerical approach is used, following the Nyquist–Shannon sampling theorem, the frequency of the effect that is simulated dictates the needed resolution of the simulation. Increasing resolution often dramatically increases computation time, so this naive solution seems not practical when simulations have to be delivered under time and budget restrictions. But with the new possibilities of GPU computing and cloud computing, the naive approach has its right to exist. Chapter 3 explores this approach with **Research Question 1**.

RQ 1

What are strategies to improve existing methods to such an extent that the resolution can be increased enough to achieve small-scale effects?

Analytical methods can describe small-scale effects in a way that is independent of simulation resolution. For complex fluid simulations, it is not possible to solve the physical model analytically. However, there is a way to combine

analytical solutions for small-scale effects with large-scale simulations. When combining the analytical solutions and the results of simulations, special attention needs to be paid to when and at which stage and with which data this happens. The small-scale effects will still need a higher sampling rate and therefore consume a lot more memory. So even if a method can save computation time, the storage and rendering of the effect may still prove difficult. Chapter 4 discusses this for **Research Question 2**.

RQ 2

How can we combine analytical solutions and simulation results in an applicable way?

The last chapter explores the idea of reducing dimensionality for the computation of the surface effects. When looking at the small-scale effects for a water surface, one idea is to solve the effect in the surface space and thus save one dimension. This idea has some obstacles. An efficient way to parametrize the surface space has to be found and a way to connect this parametrization between time steps. Especially when the topology of the surface changes during the simulation time, it gets challenging. Chapter 5 investigates this approach for **Research Question 3**.

RQ 3

How to model fine-scaled effects in 2D surface space of a 3D fluid animation?

1.2 OUTLINE AND CONTRIBUTIONS

This section presents a summary of each chapter and the author's contributions to the associated publications.

Chapter 2 - Background. This chapter summarizes basic principles and computational concepts of physically-based fluid simulation for CGI. It is based on previous work, and therefore not part of the technical contribution of this thesis. The chapter merely refreshes the necessary background for the reader.

Chapters 3, 4, and 5 are all based on publications where I was the first author. All topics were supervised and co-authored with both of the author's advisors, Bernhard Eberhardt and Daniel Weiskopf. Chapter 5 was co-authored with Stefan Reinhardt.

Chapter 3 - Distributed VFX Architecture for SPH Simulation. This chapter addresses Research Question 1 by proposing an approach to simulating and

rendering physically-based fluid effects in a VFX production environment. The approach uses a client/server architecture that is practical for distributed simulation resources and that can be seamlessly integrated into commercial three-dimensional (3D) animation packages. The fluid simulation implements Smoothed Particle Hydrodynamics (SPH). The concept of surface particles is extended by introducing "blind particles" that facilitate direct raytracing of isosurfaces. The performance of the approach is evaluated with local simulation on central processing units (CPUs) and graphics processing units (GPUs) and distributed GPU simulation. The integration into the animation package 3ds Max and the VRay raytracer is demonstrated. The usability of the VFX production pipeline is assessed by a user study with VFX professionals and animation experts.

The contributions of this chapter were presented at the World Society for Computer Graphics conference and published in the Journal of WSCG [2].

Chapter 5 - Direct Raytracing of a Closed Form Meniscus. This chapter investigates Research Question 2. It presents a direct raytracing method for implicitly described fluid surfaces that takes into account the effects of capillary solid coupling at the boundaries. The method is independent of the underlying fluid simulation method and solely based on distance fields. The method uses the closed-form solution of the meniscus shape at the fluid interface to achieve the effect of surface tension exerted by the solid object. The shape of the liquid at these boundaries is influenced by various physical properties such as the force of gravity and the affinity between the liquid and the solid material. The method generates contact angles at the boundaries without the need for computationally intensive small-scale simulation. At render time, it combines the closed-form solution for a small-scale effect with the numerical solution of a large-scale simulation. The method is applicable for any implicit representation of the fluid surface and does not require an explicit extraction of the surface geometry. Therefore, it is especially useful for particle-based simulations. Furthermore, the solution is guaranteed to yield the correct contact angle and, for certain scenarios, it delivers the correct solution throughout the interface; even in general scenarios, it yields plausible results. As an example, the proposed method is implemented and tested in the setting of an SPH fluid simulation.

The contributions of this chapter were presented at the Computer Graphics International conference and published in The Visual Computer [3].

Chapter 5 - Efficient 2D Simulation on Moving 3D Surfaces. A method is presented to simulate fluid flow on evolving surfaces, e.g., an oil film on a water surface. Given an animated surface (e.g., extracted from a particle-based fluid simulation) in 3D space, a second simulation is added on the input surface. In general, a partial differential equation (PDE) is solved on a level set surface

obtained from the animated input surface. The properties of the input surface are transferred to a sparse volume data structure that is then used for the simulation. Strategies are introduced to couple between input properties and the simulation, and conservation of mass and momentum is added to existing methods that solve a PDE in a narrow-band using the closest point method (CPM). In this way, high-resolution 2D simulations are efficiently computed on coarse input surfaces. The approach helps visual effects creators easily integrate a workflow to simulate material flow on moving surfaces into their existing production pipeline.

The author provided the initial idea and was responsible for the realization using the closest point method, the actual implementation, and the evaluation of the method. This publication was co-authored with Stefan Reinhardt, who was responsible for the mathematical model used in the simulation and its analysis. I developed all necessary software prototypes and produced all presented examples except for the examples for Fig. 5.5 and Fig. 5.6, which were produced by Stefan Reinhardt. The contributions of this chapter were presented at the Symposium on Computer Animation and published in Computer Graphics Forum [1].

1.3 REUSED AND COPYRIGHTED MATERIAL

In this thesis, material from the following copyrighted paper is partly reused with kind permission from the journal publisher.

- [2] ©2013. D. Morgenroth, D. Weiskopf, and B. Eberhardt, "Distributed VFX architecture for SPH simulation", *Journal of WSCG*, vol. 21, no. 3, pp. 163-171, 2013.

Moreover, material from a copyrighted paper is partly reused with the kind permission of Springer Nature and reused with kind permission of Springer Nature under the agreement for reuse in this dissertation (License Number 4985331333636).

- [3] ©2016. D. Morgenroth, D. Weiskopf, and B. Eberhardt, "Direct raytracing of a closed-form fluid meniscus", *The Visual Computer*, vol. 32, no. 6-8, pp. 791-800, 2016.

In addition, material from a copyrighted paper is partly reused with the kind permission of John Wiley and Sons following the license agreement for Dissertation/Thesis use (License Number 4985331008607):

- [1] ©2020. D. Morgenroth, S. Reinhardt, D. Weiskopf, and B. Eberhardt, "Efficient 2D simulation on moving 3D surfaces", *Computer Graphics Forum*, vol. 39, no. 8, pp. 27-38, 2020

CHAPTER 2

Background

The following chapter gives some background information about VFX workflows and introduces two small-scale effects. Then the necessary basics for fluid simulation are explained.

2.1 Visual Effects Workflow for Physically Based Fluid Animation

Although the laws of physics are clear, there are many different simplified models and ways to implement them. To explore different techniques for adding small-scale effects to physically based fluid simulation in a targeted manner, it is necessary to introduce requirements and constraints to find a solution that fits the concrete use case. This work puts the simulation in the context of a CGI production workflow. The solution should be applicable in the visual effects field for creating simulations that can be used to produce realistic renderings of fluids. A production pipeline for visual effects has requirements for each step to provide a defined output for the next processing stage. For example, the 3D rendering will have to provide the compositing department with a defined set of extra render elements like normal passes, velocity passes, or specular passes. New algorithms have to be implemented in a way that supports these elements. Production pipelines also place additional constraints on the system like maximum memory usage, maximum render time,

Figure 2.1: Typical building blocks for VFX production processes.

or simulation time. Design decisions must be made to meet those requirements and constraints. This section discusses a typical workflow of a CGI production.

Fig. 2.1 shows common steps for the production of a CGI shot. In the first step, "Scene Setup", all 3D models are placed in the scene and the camera is set up. This step also includes the work to create all needed 3D models for the scene, e.g., with the help of 3D modeling or scanning and the work to prepare the models for animation. For the fluid simulation part, this step includes the definition of the simulation space and setting up the initial state of the fluid body. The artists can define which objects should be considered in the simulation and the simulation parameters of the objects. Here an interesting parameter for small-scale effects could be whether a collision object for the water surface is hydrophobic like a lotus plant leaf or hydrophilic like porcelain. Another parameter could be whether the water surface is polluted with oil, and to what degree.

The next step is "Animation", where non-physically based animations are created. CG characters are often animated using a mix of motion capture techniques and traditional hand animation. Sometimes procedural animations can be used to completely replace physically based simulation for certain effects. For example, Tessendorf et al. [95] showed how to create an ocean surface with procedural wave patterns.

The "Simulation" step is the main point of interest in this work. It involves the definition of the physical model to be used and the execution of the numerical simulation. The next chapter will go into detail about both topics. It would appear that this is the only area of research relevant for improving fluid simulation with small-scale effects, but as shown later this is not the case.

The "Lighting and Shading" step consists of defining the materials and textures of the objects and the light. This work will show that small-scale effects can be added in this late stage by applying special textures to the water surface or applying shading effects like bump mapping to the surface.

This step is tightly connected to the "Rendering" step, where the images are computed. Even here, small-scale effects can be added, as will be shown in Chapter 4.

The last step is "Compositing", where the image layers are combined to the final image. Even here, small-scale effects can be added. For example, Akinci et al. [8] added screen space foam on top of an already rendered simulation.

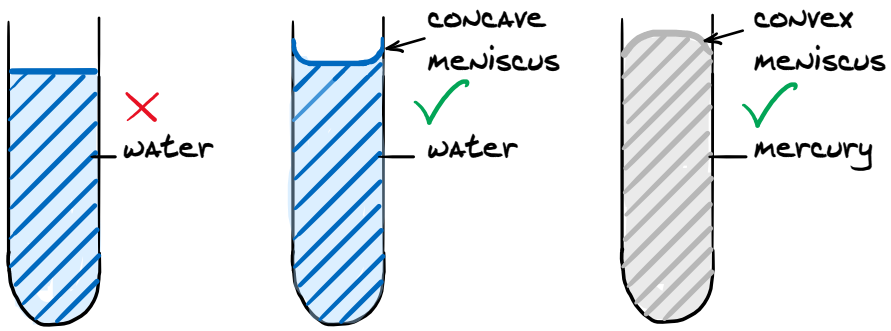


Figure 2.2: Water in a glass cylinder is not flat, it is curved at the edge. This curve is called a meniscus. Water and glass form a concave meniscus, mercury and glass form a convex meniscus.

2.2 SMALL-SCALE EFFECTS

This section describes the small-scale effects that will be added to fluid simulations in the following chapters. The first kind of effect is the meniscus effect when a fluid touches a solid object.

2.2.1 Meniscus Effect

What is a meniscus? When filling water in a glass cylinder, one would expect it to be completely flat. The gravity is the same everywhere, and the pressure is the same throughout the whole fluid body. However, in reality, almost always there is a small curve at the edge where the fluid touches the glass, as shown in Fig. 2.2. Water in glass forms a curve at the edge that is higher at the point of contact and that declines the further away it gets. This is called a concave meniscus. There are other element combinations like mercury and glass that form a convex meniscus, as shown on the right of Fig. 2.2.

Why does a meniscus occur? To understand why a meniscus occurs, the molecular level of fluids should be examined [5]. Fig. 2.3 depictures the molecules for water that consists of two hydrogen atoms and one oxygen atom. The water molecule is a dipole molecule, that means although its net electronic charge is zero, it has a slightly positive charge on the hydrogen side and a slightly negative charge on the oxygen side. Oxygen is extremely electro-negative, making it more likely that a shared electron is on the oxygen side than on the hydrogen side. This polarity is why water molecules attract each other. This attraction between the molecules is called hydrogen bonds. Fig. 2.4 visualizes these attraction forces between the water molecules. These forces are also called cohesive forces.

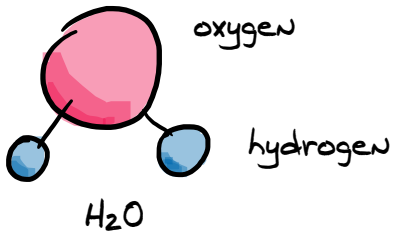


Figure 2.3: A water molecule consists of one oxygen atom and two hydrogen atoms.

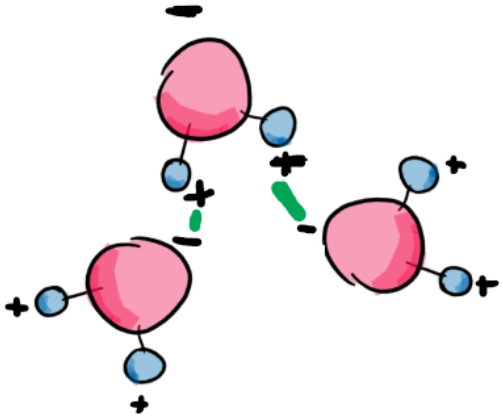


Figure 2.4: Water molecules are polar and have an end with more negative charge and one end with a more positive charge. This causes the hydrogen bonds, an attraction force (green) between the molecules.

Water molecules are not only attracted to each other. They are attracted to other polar molecules, too. Fig. 2.5 shows water in a glass. The attraction forces to other materials are called adhesive forces. Glass is made of silica sand. It consists of a silica oxygen lattice (SiO_2). For every silicon atom, there are two oxygen atoms. The electronegativity difference between oxygen and silicon is greater than the one between hydrogen and oxygen. The adhesive force between the water molecules and the silica molecules is greater than the water molecules' cohesive forces. As a consequence, the water molecules are attracted more to the glass container than to each other. When kinematic energy in the water pushes a water molecule slightly higher at the boundary, it will stick to the surface due to the adhesive forces and pull other water molecules with the cohesive forces. This phenomenon causes the curved meniscus at the edge [27]. When there is a cylinder of glass with a small diameter, these adhesive forces have a relatively more dominant surface and the water will slowly rise up the cylinder. This effect is called the capillary effect.

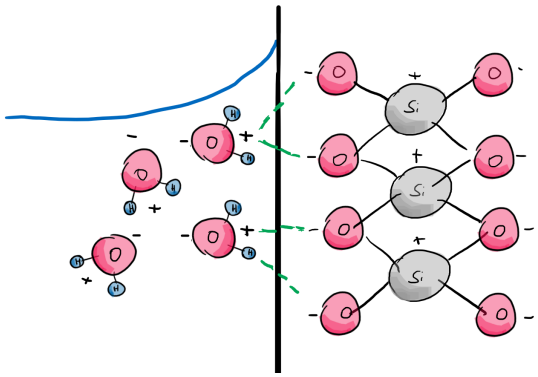


Figure 2.5: For materials as glass with silicate molecules, the adhesion forces between the water molecules and the glass are greater than the cohesive forces between the water molecules.

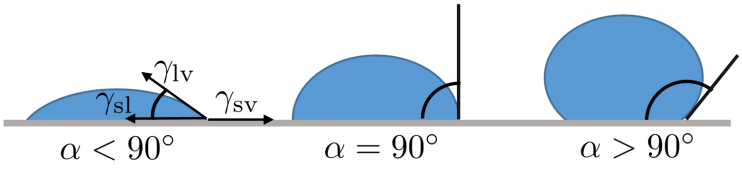


Figure 2.6: Illustration of contact angles formed by sessile liquid drops on smooth homogeneous solid surfaces of varying material (left to right: high wettability to low wettability). Image inspired by Yuan and Lee [108].

The contact angle between fluid and solid is an important property of drops on surfaces and largely affects their visual appearance. According to Young [105], the contact angle is described by the mechanical equilibrium of the fluid drop under the influence of three interface tensions:

$$\gamma_{lv} \cos \alpha_Y = \gamma_{sv} - \gamma_{sl} \,, \tag{2.1}$$

where γ_{lv},γ_{sv}, and γ_{sl} are the liquid–vapor, solid–vapor, and solid–liquid interface tensions. The contact angle is denoted α_Y. Eq. 2.1 is referred to as Young's equation [56].

The effect of the contact angle is well visible when one considers droplets on various flat surfaces having different properties. Hydrophilic materials (high wettability) form small contact angles ($<$ 90 degrees), whereas hydrophobic materials (low wettability) lead to large contact angles, as illustrated in Fig. 2.6.

2.2.2 Iridescence Effect

The second kind of small-scale effect used later is the shimmering rainbow color effect that occurs on oil films on water. The pearly colors change when

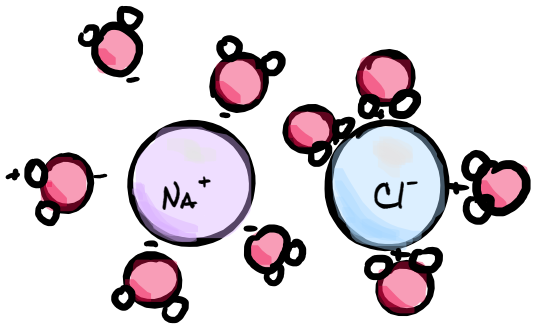


Figure 2.7: Charged chemical compounds like sodium chlorid (NaCl) dissolve well in water as the positively and negatively charged ions are surrounded by the polar water molecules.

the angle of view or the angle of light changes. This color effect is called iridescence effect and is caused by interference effects when light rays are reflected in very thin optical films, where the thickness is in the magnitudes of the light's wavelengths. This small thickness is why this effect is considered a small-scale effect in the context of this work.

Why does oil and water not mix up well? Because of the polar nature of water molecules, water is a good solvent for most chemical compounds. Salt, for example, is a chemical compound of sodium (Na) and chloride (Cl) with positive and negative charges. Positive and negative ends of the water molecules easily attach to those positive and negative ions of salt as shown in Fig. 2.7. Generally, all charged chemical compounds solve well in water. They are called hydrophilic. In contrast, oil is a hydrocarbon which does not consist of charged compounds. There are no positive and negative ions where the ends of the water molecules could attach. These materials are hydrophobic materials and do not dissolve in water. If one drops oil in water it will not mix or dissolve. Since oil has a lower density than water, it will float on top of the water.

How is the color effect created on the oil film? Oil and water have different optical properties. They both are transparent and light can cross the material, but the index of refraction (IoR) of the two materials is different. Because of the different IoRs, light is partially reflected when it crosses the boundary between those two materials.

When light hits an oil film, it is partly reflected and partly refracted. The refracted ray is reflected at the backside of the oil film and refracted back. There can also be rays that are reflected at both interfaces of the layer multiple times before they leave the thin film.

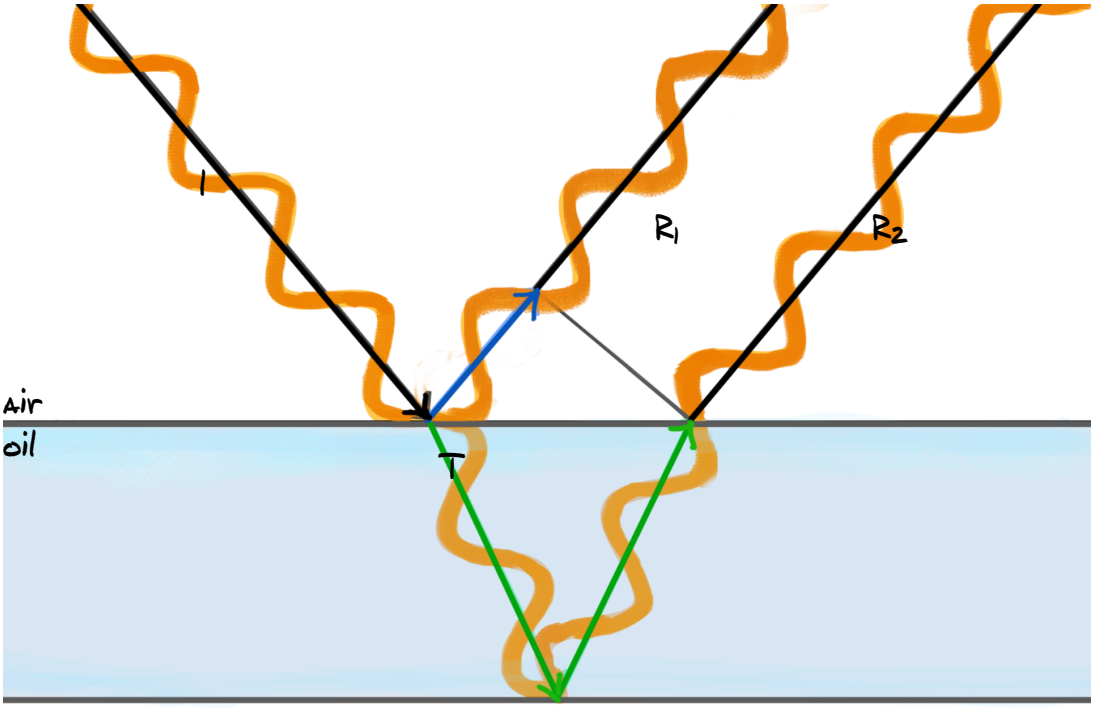


Figure 2.8: A light ray I comes from the upper left and hits a transparent thin film layer where it is then both refracted into ray T and reflected into ray R_1. The first reflection R_1 and the second reflection R_2 from the inner backside have an offset that is the difference between the green and blue path. This wave phase offset then causes the interference effects when the two waves are added.

When one sees white light there are many rays with overlapping wavelengths. The receptors in our eyes add them up and perceive them as one color. Fig. 2.8 visualizes a ray which is partly refracted and reflected, there are phase differences in the waves between the first reflection and the inner reflection. When such two waves are added up, this causes interference. Depending on the light's wavelength, the traveled distance causes a phase difference and, consequently, two rays could either add up or cancel each other. As a result, some colors are stronger than others. This eventually produces the different colors that one sees on thin oil films.

Wu et al.[101] derive a full-spectrum multilayer film interference method to render the interference effect based on the interference theory of multilayer films [17].

2.3 LENGTH SCALES IN FLUID MECHANICS

Depending on which length scale is considered, many different phenomena in the field of fluid mechanics can be found. Fig. 2.9 assigns some phenomena to the length scales. The use-case and the length scale determine which physical model and numerical methods best simulate the behavior.

For generic fluid mechanics, the Knudsen number gives insight into which approach best describes a phenomenon. The Knudsen number measures the ratio of the mean free path of the molecules to a geometric reference length of the effect that is described, e.g., the radius of a swirl that is computed. Small Knudsen numbers relate to large scales. Depending on the Knudsen number's magnitude, different governing equations should be used [6]. As a consequence, to model phenomena with different Knudsen numbers, there are different equations for the different phenomena.

In the case of micro-scale with Knudsen numbers larger than 10, the molecules' motion and forces are often directly calculated with the Newtonian equation of motion. Flow is caused by the movement of a large number of molecules. The undirected Brownian molecular movement is superimposed on the actual flow movement. With increasing density, in addition to the free movement of molecules, collisions play a role.

In larger scales with Knudsen numbers between 10^{1} and 10^{-1}, mesoscopic methods are typically used that exploit particle-based simulations of chunks of fluid atoms to capture adhesion and capillary forces adequately. The governing equations in this scale are typically the Burnett equations. A good introduction to the Burnett equations can be found in the work from Agarwal et al. [6].

In the larger macroscopic length scale, continuum mechanics can be used by solving the Navier-Stokes equations following the fundamental physical laws of conservation of mass, conservation of momentum, and conservation of energy. For Knudsen numbers between 10^{-1} and 10^{-3}, Navier-Stokes equations with slip conditions should be used [83]. If the Knudsen number is smaller than 10^{-5}, viscosity and thermal conductivity are negligible and therefore the Euler equations can be used.

From the phenomena that are discussed in this work, the one with the highest Knudsen number is the interference pattern of an oil film on a water surface. The difference of thickness that causes the pattern is in the nanometer scale [80]. As water molecules (in liquid state) are in direct contact with 4 to 5 other water molecules [97], the mean free path of the molecules for liquid water can be set to be the average distance of two oxygen molecules, which is 2,85A and as such in the magnitude of 10^{-10}m [30]. This still leads to a Knudsen number

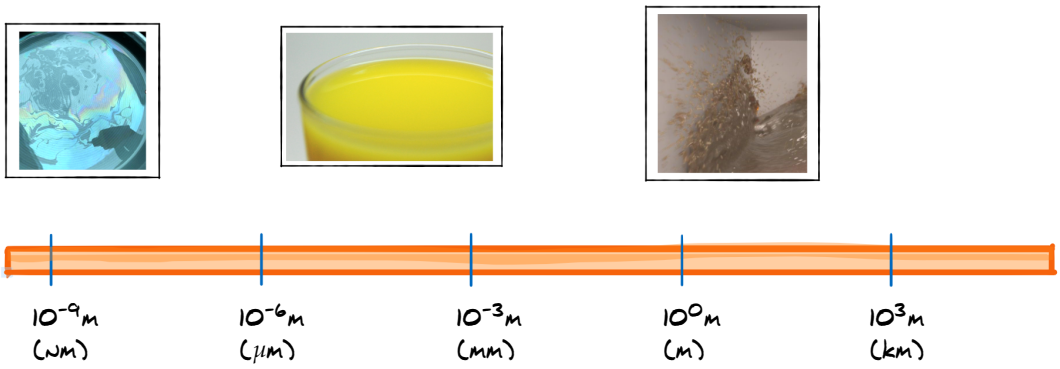


Figure 2.9: Different length scales of fluid phenomena. In the nanometer length scale on the left, there are the soap film light interference effects. In the mid-scale are effects like the meniscus effect when glass touches a boundary object. In the larger scales, there is the fluid motion.

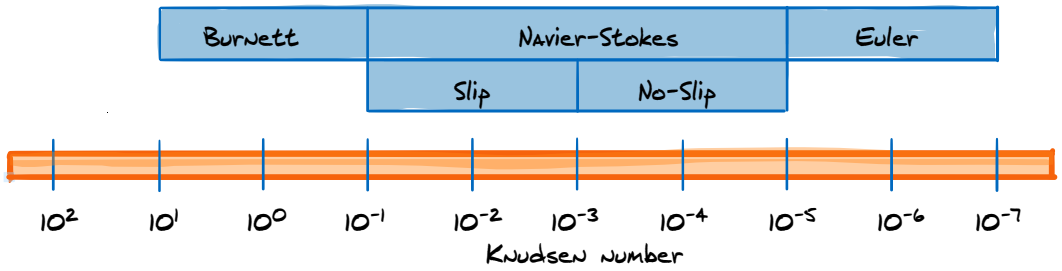


Figure 2.10: Different Knudsen numbers require different governing equations [83].

of 10^{-1}. Consequently, to enhance large-scale simulations with small-scale effects, the correct governing equations are the same as those for the mid-scale to large-scale, i.e., the Navier-Stokes equations.

2.4 NAVIER-STOKES EQUATIONS

The governing equations of fluid phenomenons with Knudsen numbers between 10^{-1} and 10^{-3} are the Navier-Stokes equations with slip conditions where the fluid body is treated as a continuum. An introduction to these equations can be found in the work from Batchelor et al. [14]. These equations describe how the fluid velocity $\mathbf{u}$, pressure p, and density ρ of a moving fluid are related.

This work mainly uses the version of the Navier-Stokes equations for incompressible fluids. They can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla \cdot (\nu \nabla \mathbf{u}) - \frac{1}{\rho}\nabla p + \mathbf{f}, \tag{2.2}$$

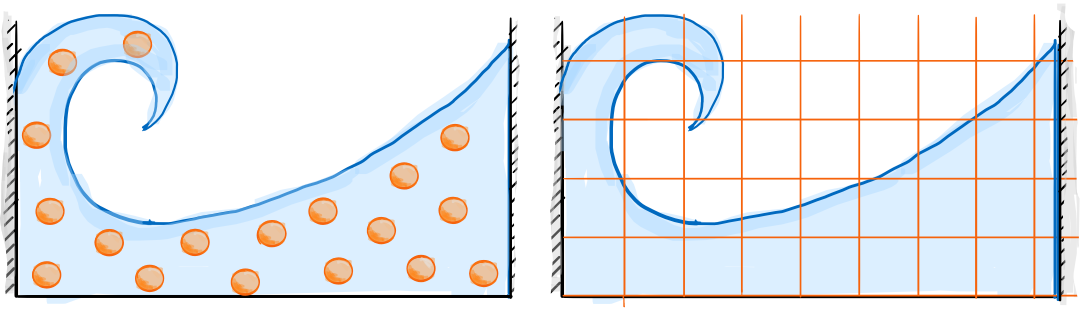


Figure 2.11: On the left, the Lagrangian view of a fluid is visualized where the individual chunks of fluid are tracked. On the right, you see the Eulerian view where the properties are tracked in static volumes in space.

$$\nabla \cdot \mathbf{u} = 0. \tag{2.3}$$

The Navier-Stokes equations describe the velocity change over time (on the left) through the following four physical processes / forces: advection, diffusion, pressure, and external body forces.

The advection term is mathematically given by $-(\mathbf{u} \cdot \nabla)\mathbf{u}$. It describes the force exerted on a particle of fluid by other particles surrounding it. The term $\nabla \cdot (\nu \nabla \mathbf{u})$ is the velocity diffusion. It describes how fluid motion is damped due to kinematic viscosity ν. Highly viscous fluids like honey stick together, where as low-viscosity fluids like air flow freely. $\frac{1}{\rho}\nabla p$ is the pressure term. Fluid flows in the direction of the largest change in pressure. The external forces $\mathbf{f}$ are e.g., gravity or forces that are exerted by collision objects. The continuity equation $\nabla \cdot \mathbf{u} = 0$ for constant density flow ensures the conservation of mass by requiring a divergence free velocity field.

To adequately simulate the behavior of fluids, these partial differential equations (PDEs) have to be solved. The different numerical approaches can be divided in two categories: Lagrangian or Eulerian. Fig. 2.11 depicts the two approaches. In the Lagrangian description of liquid flow, discrete points carry field quantities and move with the fluid. In the Eulerian description, a control volume tracks the properties like pressure, and acceleration describes them as dynamic fields that change over time.

Using the Navier-Stokes equations and a Lagrangian or Eulerian numerical method, fluid effects for fluid phenomenons with a Knudsen number between 10^{-1} and 10^{-5} can be described. In this work both approaches are used. While in Chapter 3 a Lagrangian system is described, Chapter 5 applies an Eulerian approach. Both approaches are briefly explained in the following two sections.

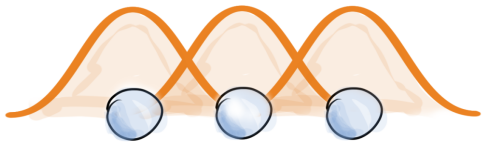


Figure 2.12: Three SPH particles near each other with a graphical representation of the kernel function. To spatially calculate the value of a property the kernel functions of each particle are used.

2.5 SMOOTHED PARTICLE HYDRODYNAMICS – A LAGRANGIAN METHOD

The SPH method was first introduced by Gingold and Monaghan [33] in 1977. An in-depth overview of the state-of-the-art can be found in Koschier et al. [54].

2.5.1 Spatial Discretization

Particle-based methods like SPH are more intuitive than grid-based methods because every particle represents a certain amount of the fluid body mass. The problem with a point-based approach for fluids is that the space between the particles is not defined. The particles as shown in Fig. 2.12 represent sample points that carry quantities like mass, temperature, density, or velocity. Thus, the particles are helpers to sample the continuous fluid and keep track of various quantities. The resolution is much coarser than the actual molecules of the fluid and is typically defined beforehand. So, inside the fluid between the particles is empty space. To estimate a quantity like temperature or pressure in this space between the particles, the fundamental idea of SPH is to take the average value of all neighboring particles in this area in a way that each particle's value is weighted with a kernel function that decreases with distance. Thus, nearby particles contribute more, distant particles contribute less. Fig. 2.12 depicts the weighting functions around the particles.

To simplify the notation for the next section, the position $\mathbf{x}$ is defined as the position of evaluation. In the case that a quantity is calculated for a certain particle i, the position is denoted as $\mathbf{x}_i$. All neighbor particles j in the region of influence that is controlled with the smoothing length h contribute to a quantity. The position $\mathbf{x}_j$ is defined as the position of a neighboring particle j. Further, r is defined as the distance between $\mathbf{x}$ and $\mathbf{x}_j$.

$$r = \|\mathbf{x} - \mathbf{x}_j\|. \tag{2.4}$$

The main building blocks of the SPH method are the kernel functions W. Examples of these functions are shown in Fig. 2.13. They are weight functions that account for the distance r and a smoothing length h that is defined beforehand.

To further simplify the equations, the following notation is used:

$$W(\|\mathbf{x}_i - \mathbf{x}_j\|, h) = W(r, h) = W_{ij}. \tag{2.5}$$

In SPH, a quantity A is evaluated at a position $\mathbf{x}$ by summing up the weighted contributions of its neighboring particles j:

$$A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W_{ij}, \tag{2.6}$$

where m_j is the mass of particle j, ρ_j its density, and W_{ij} the smoothing kernel with smoothing length h.

One important property is the density of the fluid. For every particle i, it can be computed by

$$\rho_i = \sum_j m_j W_{ij}. \tag{2.7}$$

The smoothing kernels used in SPH are zero outside the smoothing length h, that means they have compact support $\Omega(\mathbf{x}, h)$:

$$W_{ij} = 0 \text{ if } \|\mathbf{x}_i - \mathbf{x}_j\| > h. \tag{2.8}$$

This property implies that for the computation of quantities only the neighboring particles need to be considered, i.e., all particles that are within the smoothing length h. All particles that are further away than h do not contribute. The kernel functions are non-negative and they are normalized such that the integral inside the compact support sums up to 1:

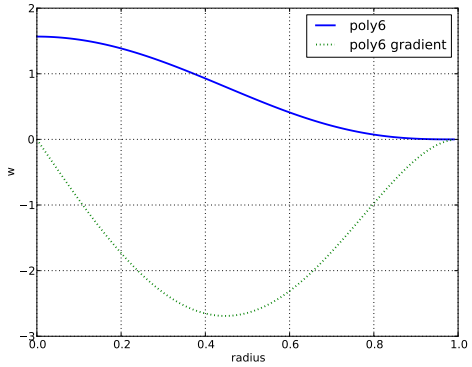
$$\int_{\Omega(\mathbf{x}_i, h)} W(\|\mathbf{x}_i - \mathbf{x}\|, h) dv(\mathbf{x}) = 1, \tag{2.9}$$

where dv denotes the volume integration variable corresponding to $\mathbf{x}_j$.

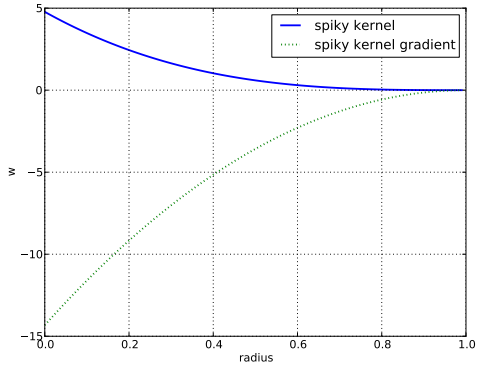
Fig. 2.13 shows typical kernel functions for SPH from Mueller et al. [67]. These functions were also used for the implementation in this work.

The poly6 kernel as shown in Fig. 2.13a is mainly used for the calculation of the density and other scalar quantities. It is calculated with:

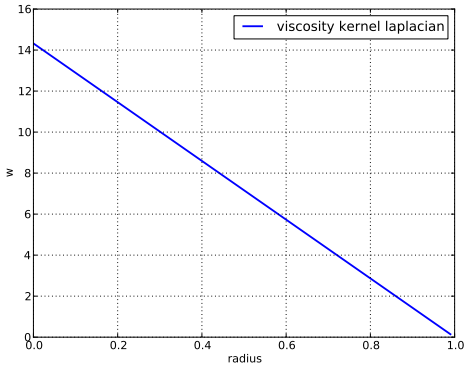
$$W_{\text{poly6}}(r, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$



(a) The poly6 kernel function



(b) The spiky kernel function



(c) The Laplacian of the viscosity kernel function

Figure 2.13: Typical kernel functions that are used in SPH.

The spiky kernel (Fig. 2.13b) is used for the pressure computations and is written as:

$$W_{\text{spiky}}(r,h) = \frac{15}{\pi h^6} \begin{cases} (h-r)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

For the viscosity computation, the Laplace of the viscosity kernel (Fig. 2.13c) is used.

$$W_{\text{viscosity}}(r,h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \leq r \leq h \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

The fluid properties needed to solve the Navier-Stokes equations are either scalar fields or vector fields. Those quantities A are functions in space and time. The rate of change of such a quantity is the material derivative $\frac{D}{Dt}$. When the fluid moves, those quantities are then moved along with the particles. In

the Lagrangian setting, this simplifies the material derivative to

$$\frac{DA}{Dt}(\mathbf{x}(t),t) = \frac{\partial A}{\partial t}(\mathbf{x}(t),t). \tag{2.13}$$

2.5.2 Discretization of Differential Operators

To solve the Navier-Stokes equations, the derivatives of velocity and pressure are necessary. For the derivative of an arbitrary quantity, the formula in Eq. 2.14 can be used. Derivatives are calculated similar to the values of a field quantity, but for derivative values the derivative of the kernel function is used.

$$\nabla A(r) \approx \sum_j m_j \frac{A_j}{\rho_j} \nabla W_{ij}. \tag{2.14}$$

This can be extended to vector-valued quantities and other typical differential operators:

$$\nabla \mathbf{A}(\mathbf{x}) \approx \sum_j \frac{m_j}{\rho_j} \mathbf{A}_j \otimes \nabla W_{ij}, \tag{2.15}$$

$$\nabla \cdot \mathbf{A}(\mathbf{x}) \approx \sum_j \frac{m_j}{\rho_j} \mathbf{A}_j \cdot \nabla W_{ij}, \tag{2.16}$$

$$\nabla \times \mathbf{A}(\mathbf{x}) \approx -\sum_j \frac{m_j}{\rho_j} \mathbf{A}_j \times \nabla W_{ij}. \tag{2.17}$$

For the Laplace operator, the second derivative is used:

$$\nabla^2 A(r) \approx \sum_j \frac{m_j A_j}{\rho_j} \nabla^2 W_{ij}. \tag{2.18}$$

These formulas for derivatives of quantities can now be used for the Navier-Stokes equations from Eq. 2.2.

The force $\mathbf{f}^v$ from viscosity can be calculated with:

$$\mathbf{f}^v = \mu \sum_j \frac{m_j(\mathbf{u}_j - \mathbf{u}_i)}{\rho_j} \nabla^2 W_{ij}, \tag{2.19}$$

where μ is the dynamic viscosity that is considered constant. The gravity force $\mathbf{f}^g$ is:

$$\mathbf{f}^g = m_i \mathbf{g}. \tag{2.20}$$

Having viscosity and gravity equations, the missing part now is the pressure. Koschier et al. [54] give an overview of possible pressure solvers for SPH.

2.5.3 Time Integration

The Navier-Stokes equations (see Eq. 2.2) are a set of coupled nonlinear PDEs. The spatial discretization with the SPH approach as well as the simplifications by constant temperature and incompressibility reduce the PDEs to a system of ordinary differential equations (ODEs). These ODEs can be solved by discretizing time. Given an initial configuration of particles described by positions $\mathbf{x}_j(t_0)$ and velocities $\mathbf{u}_j(t_0)$, the temporal changes of positions and velocities are generally given by

$$\frac{d\mathbf{u}}{dt}(t) = \mathbf{a}(t), \tag{2.21}$$

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{u}(t). \tag{2.22}$$

The acceleration $\mathbf{a}$ is defined by the forces for pressure, viscosity, and external forces divided by the particle mass.

This typical system of ODEs can be solved with time integration. Using a fixed time step Δt, the new position and velocity can be calculated with a first-order Euler scheme:

$$\mathbf{u}_j(t + \Delta t) = \mathbf{u}_j(t) + \Delta t \mathbf{a}_j(t), \tag{2.23}$$

$$\mathbf{x}_j(t + \Delta t) = \mathbf{x}_j(t) + \Delta t \mathbf{u}_j(t). \tag{2.24}$$

2.5.4 PCISPH

One widely used approach for the pressure solver is the Predictive–Corrective Incompressible SPH (PCISPH) approach from Solenthaler et al. [85]. The main idea is to predict the resulting density ρ_i^* using the current position and velocity as if there was no pressure force. Then, from the predicted density, the density error $\rho_0 - \rho_i$ is derived, where ρ_0 is the rest density of the fluid. This predicted density error can then be transformed to a predicted pressure p_i that is needed to eliminate the error, using a stiffness constant:

$$p_i = k^{\text{PCI}}(\rho_0 - \rho_i^*). \tag{2.25}$$

This is a state equation where the stiffness constant k^{PCI} is not defined by the user but precomputed from a template particle surrounded by an optimally sampled neighborhood and derived from the fact that the pressure must induce an acceleration so that the particles reach their rest density at the next time step $t + \Delta t$. The derivation of the stiffness constant and its comparison with stiffness values in other SPH approaches are discussed in the report by Koschier et al. [54]. The stiffness constant is given by:

$$k^{\text{PCI}} = \frac{-\rho_0^2}{2\Delta t^2 m^2 (-\sum_j \nabla W_{ij} \cdot \sum_j \nabla W_{ij} - \sum_j \cdot \nabla W_{ij})}. \tag{2.26}$$

The predicted pressure is then iteratively refined by using it to predict better velocities and positions that again result in predicted density errors and pressures. This loop of predicting the pressure and refining it is stopped as soon as the maximum density error reaches a certain error threshold ϵ. The force created by the pressure is then calculated with:

$$\mathbf{f}^p = -m^2 \frac{2\rho_i^*}{\rho_0^2} \sum_j \nabla W_{ij}. \tag{2.27}$$

With the formulas for the integration, all needed parts to simulate a fluid with SPH are now in place. The basic algorithm is the following.

Algorithm 1: PCISPH Algorithm

```
calculate k^PCI
define ρ_0
while in simulation loop do
    for every particle i do
        compute forces for viscosity (Eq. 2.26) (Eq. 2.19), gravity
         (Eq. 2.20), and external forces
        initialize pressure p and pressure force f^p with 0
    end
    do
        for every particle i do
            predict position x_i^* at (t + Δt) (Eq. 2.24)
            predict velocity u_i^* at (t + Δt) (Eq. 2.23)
        end
        for every particle i do
            predict density ρ_i^* (Eq. 2.7)
            predict density error (ρ_0 − ρ_i^*)
            update pressure p_i += k^PCI (ρ_0 − ρ_i^*)
        end
        for every particle i do
            compute pressure force f^p (Eq. 2.27)
        end
    while (ρ_0 − ρ_i^*) > ε;
    for every particle i do
        compute position x_i at (t + Δt) (Eq. 2.24)
        compute velocity u_i at (t + Δt) (Eq. 2.23)
    end
end
```

If a too large time step is chosen, the simulation gets unstable, if the time step is too small, the simulation takes too long. The sweet spot can be determined using a heuristic based on the Courant–Friedrichs–Lewy (CFL) condition. The

CFL condition describes an upper limit of the time step for the convergence of numerical solvers for differential equations. The condition states:

$$\Delta t \leq \lambda \frac{h}{||\mathbf{u}^{\max}||}, \tag{2.28}$$

where λ is an arbitrary scaling parameter, h is the particle size, and $\mathbf{u}^{\max}$ the velocity of the fastest particle. The CFL condition brings the particle size in relation to the largest possible time step. In the topic of small-scale effects, this has direct consequences when the particle size needs to be reduced to capture smaller effects. The computation time increases not only because of the larger amount of particles needed to fill the same volume but also because smaller time steps are necessary.

This leads to one crucial topic – the computation time with increasing particle amounts and smaller time steps. The problematic area of the SPH approach is that for every quantity of a particle that has to be calculated, the corresponding quantities of all neighboring particles have to be queried.

2.5.5 Acceleration Structures

Each particle obtains its properties based on the properties and distances to its neighbors. Naive distance calculation for each particle against all other particles would lead to complexity of $O(n^2)$. Only neighbors that are closer than the smoothing length h contribute to the properties of the particle. Therefore, an algorithm that returns only the neighbors within the smoothing length h greatly improves the complexity. Finding the neighbors and looping over all of them is the most time-consuming step. Therefore, it is necessary to implement some acceleration structures for the neighborhood search. To accelerate neighbor queries, the most common solution is to subdivide the space into regions. This approach is called spatial subdivision. In many SPH simulation frameworks, the smoothing length h is constant for all particles. The query only accounts for neighboring particles within the smoothing length. This makes a regular grid with side length h the natural choice as an acceleration structure as depicted in Fig. 2.14. For every particle, the particles that are in the same cell and those of the neighboring cells have to be considered.

To find all particles of a cell, different implementations can be used. One solution is to put variable-length arrays for each cell into a hash map that is indexed by the cell coordinates. Then each particle is added to the corresponding array depending on its cell location. This ideal solution is hard to achieve with current GPU frameworks since neither hash maps nor variable-length arrays are available. Ihmsen et al. [42] compared different neighbor search data structures for multi-core architectures with shared memory and found that using a regular grid for neighborhood search with zCurve sorting to be the

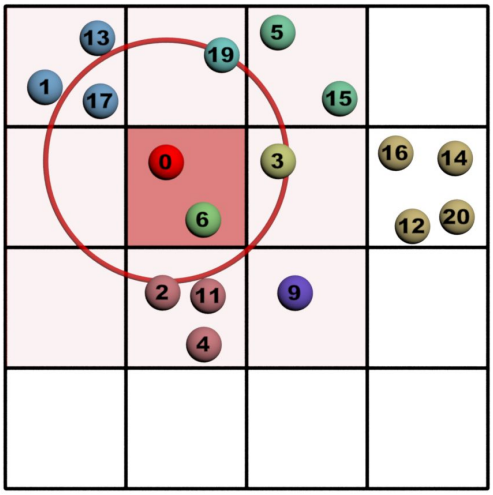
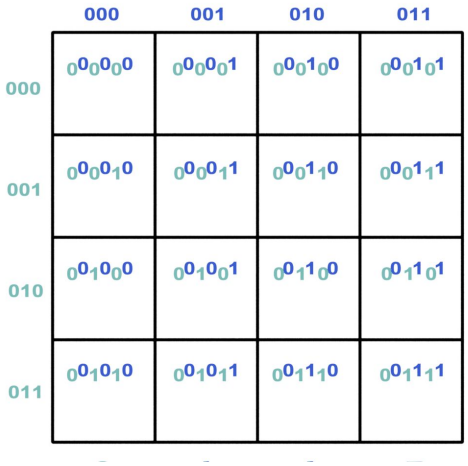
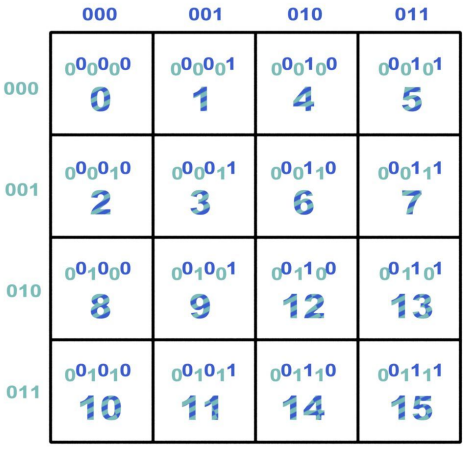


Figure 2.14: Neighborhood of particle 0.

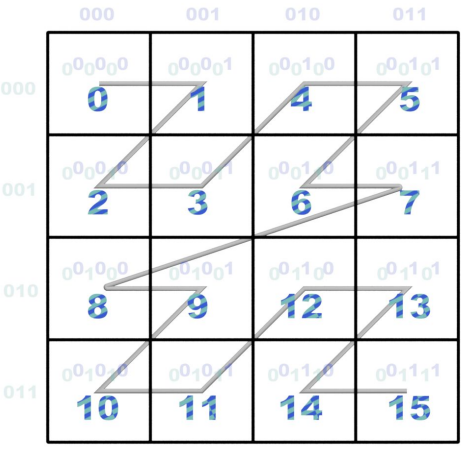
fastest solution for parallel implementations. This work follows the approach used by Goswami et al. [34] using a space-filling curve from Morton [66]. Morton curves map three-dimensional space to one dimension with a curve that travels through space in a "Z" shape. Therefore, Morton curves are also called z-curves. The Morton z-value of a point in multiple dimensions is computed by nesting the binary representations of its coordinate values as shown in Fig. 2.15. By this approach, every cell of a regular grid gets one Morton coordinate instead of x,y, and z coordinates. Points in 3D space which are close to each other in the N-dimensional space have Morton numbers that are close to each other, too. This creates beneficial cache properties when used in computations that query cells that are close to each other in space. Particle attributes can now be stored in one-dimensional arrays as shown in Fig. 2.15d. The arrays are sorted based on their Morton z-value. To speed up the query for all particles that belong to a cell, a helper array is constructed that has an entry for each non-empty cell with the index of the first particle in the sorted particle array with that corresponding z-value and the number of particles in the cell. To query all particles that are close to a point in space, the algorithm calculates the z-values for the corresponding grid cell and the neighbor cells. The particles of a cell are found with a binary search on the z-values of the helper array. Then with the index of the starting particle and the number of particles that belong to the cell, all particles of that cell can be queried.



(a) The binary form of the coordinates of the grid cells are interleaved



(b) the resulting decimal number is the new z coordinate of the cell



(c) the connected coordinates form a space-filling curve

(d) particles are given their z coordinate and sorted accordingly

Figure 2.15: Particles sorted according to space-filling Morton code curve.

2.6 PROJECTION METHOD – AN EULERIAN METHOD

In contrast to the Lagrangian SPH approach, where fluid elements are tracked with particles that flow with the fluid, the Eulerian approaches keep track of properties at fixed locations in space as a function of time. The main difference is that the advection or transport of quantities of fluid does not happen implicitly when the fluid particles travel and bring their assigned quantity with them. Instead, an extra step is needed where the quantities at the fixed locations are moved. In fact, scalar quantities $A(\mathbf{x},t)$ are functions of space and time and a velocity field $\mathbf{u}(\mathbf{x},t)$ of the fluid. This results in a simple advection equation with the material derivative:

$$\frac{D}{Dt}A = \frac{\partial A}{\partial t} + \mathbf{u}\cdot\nabla A. \tag{2.29}$$

When setting the equation equal to zero, all the changes of the property happen because the quantity is traveling along with the velocity of the fluid and is not changing in respect to the Lagrangian view of one fluid particle. By this advection step, the velocity itself is also transported. Transporting the velocity field with itself may feel confusing, but in the end, it is also simply a property that is transported along with the fluid as density, pressure, or any other quantity.

2.6.1 Incompressibility

Ensuring incompressibility is another topic where the Lagrangian point of view is more intuitive. With a particle representation of the fluid it is obvious when the density of the fluid changes, as when there are particles with equal mass, the particles in this area will clutter together in areas with higher density and be wider apart in areas with lower density. Ensuring incompressibility then means: "make sure the particles are the same distance apart in all areas in the next time step." In an Eulerian system, it is not that easy to describe the problem. Here, the divergence operator can be used:

$$\nabla\cdot\mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}. \tag{2.30}$$

The divergence operator for a vector field measures how the vectors are diverging or converging. More intuitively, if the divergence of a cell is positive, more material is flowing out of that cell and if the divergence is negative, more material is flowing into the cell. With this in mind, the incompressibility condition of the Navier-Stokes equation (Eq. 2.3) becomes more intuitive:

$$\nabla\cdot\mathbf{u} = 0. \tag{2.3 revisited}$$

This means, in each cell, the amount of fluid that goes into the cell is the same amount that goes out of the cell and the amounts sum up to zero. The task for a fluid solver here is to make sure that in the next time step, the velocity field is again divergence-free. If there is a negative divergence for a cell, this essentially means that there is more fluid flowing into the cell than out, which then increases the pressure. Using this pressure, now a pressure force can be calculated that is pointing in the direction of the pressure gradient that will create the needed velocity change for the next time step $t + \Delta t$ to transform a diverging velocity field $\mathbf{w}$ into a divergence-free velocity field $\mathbf{u}$. Algorithms that solve the Navier-Stokes equation (Eq. 2.2) with an Eulerian approach often split the advection, external forces, and the pressure part into subtasks that are then solved sequentially.

First, using advection, viscosity, and body forces, a velocity field $\mathbf{w}$ is constructed that does not satisfy continuity, but then is corrected using a pressure gradient. The Navier-Stokes equations in discretized form can then be split into a part with acceleration $\mathbf{a}_{\text{AVE}}$ due to advection, viscosity, and external forces, that leads to an intermediate velocity $\mathbf{w}(t + \Delta t)$, and a part due to the pressure part.

$$\mathbf{w}(t + \Delta t) = \mathbf{u}(t) + \Delta t \cdot \mathbf{a}_{\text{AVE}} \tag{2.31}$$

$$\mathbf{u}(t + \Delta t) = \mathbf{w}(t + \Delta t) - \Delta t \frac{1}{\rho} \nabla p. \tag{2.32}$$

2.6.2 Conjugate Gradient Solver

In this last step, the divergence-producing part of the velocity is "projected out". Therefore, this step is called the pressure projection step. A method that solves the Navier-Stokes equations using this pressure projection was adapted to fluids for computer graphics by Stam [88]. This method can also be enhanced to work in parallel on multi-GPU systems [9]. Stam shows how to modify the Navier-Stokes equation using a projection operator $\mathbf{P}$. The starting point is the Helmholtz-Hodge decomposition, which states that a vector field $\mathbf{w}$ can be composed into a divergence free vector field $\mathbf{u}$ and the gradient of a scalar field q:

$$\mathbf{w} = \mathbf{u} + \nabla q, \tag{2.33}$$

where $\nabla \cdot \mathbf{u} = 0$. The operator $\mathbf{P}$ projects the velocity field $\mathbf{w}$ to a divergence free velocity field $\mathbf{u}$:

$$\mathbf{u} = \mathbf{P}(\mathbf{w}) = \mathbf{w} - \nabla q. \tag{2.34}$$

The operator is defined when multiplying Eq. 2.33 with ∇, leading to the Poisson equation:

$$\nabla \cdot \mathbf{w} = \nabla^2 q. \tag{2.35}$$

Stam applies this operator to the Navier-Stokes equations, Eq. 2.2 and Eq. 2.3, on both sides with using the fact that $\mathbf{Pu} = \mathbf{u}$ and $\mathbf{P}\Delta p = 0$. This creates modified Navier-Stokes equations in one single equation:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(-(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}). \tag{2.36}$$

To find the required pressure, time is discretized in time steps Δt and space with a regular grid with spacing Δx. Eq. 2.32 can be rewritten as:

$$\nabla p = \frac{\rho}{\Delta t}\mathbf{w}(t + \Delta t) - \frac{\rho}{\Delta t}\mathbf{u}(t + \Delta t). \tag{2.37}$$

With this substitution and applying the "∇" operator on both sides, and exploiting $\nabla \cdot \mathbf{u} = 0$, Eq. 2.37 can be rewritten as:

$$\nabla^2 p = \frac{\rho}{\Delta t}(\nabla \cdot \mathbf{w}) := b. \tag{2.38}$$

Please note that when setting $q = \frac{\Delta t}{\rho}$, this resembles Eq. 2.35. On a regular grid, finite differences can be used to solve the derivatives. The Laplacian is approximated with central differences:

$$\nabla^2 p \approx \frac{6p - p_{x-1} - p_{x+1} - p_{y-1} - p_{y+1} - p_{z-1} - p_{z+1}}{\Delta x^2}. \tag{2.39}$$

This can be written as a large system of linear equations

$$A\mathbf{x} = \mathbf{b}, \tag{2.40}$$

where the vector $\mathbf{x}$ represents all unknown pressure values of the grid cells, the vector $\mathbf{b}$ holds the scaled negative divergences of each fluid cell, and a large coefficient matrix A holds the coefficients for the Laplacian operator.

There are many solvers for equations of that form. One widely used solver is the preconditioned conjugate gradient solver with a Jacobi preconditioner which is part of the software development kit of the Houdini software package. A thorough introduction to the conjugate gradient method was written by Jonathan Shewchuk [81]. The basic idea of using a gradient descent method for the equation $A\mathbf{x} = \mathbf{b}$ is that this is equivalent to find a minimum of the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + c. \tag{2.41}$$

If A is a symmetric positive definite matrix, this function has its minimum when the derivative is zero. The above function is designed that its derivative is the Poisson equation $A\mathbf{x} = \mathbf{b}$. Gradient descent methods find the minimum of the equation by starting at an arbitrary point and then iteratively descending a step in the direction of the gradient closer to the solution. The conjugate gradient method is a variant of the gradient descent method that improves the number of needed steps by choosing the search directions in a way that they are orthogonal to each other. This means for each search direction you find the optimal step that needs no correction in that direction in subsequent steps. As a result, the method is guaranteed to find the exact solution after n steps for an $n \times n$ matrix. If fewer than n steps are taken, then the rate of convergence is related to the condition number κ of A, which is defined as the ratio of the largest and smallest eigenvalue of A. The rate of convergence is defined as:

$$||e_{(i)}||_A \leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)^{-1} ||e_{(0)}||_A. \tag{2.42}$$

In other words, if the condition number of matrix A is low, then the method converges faster. For an ill-conditioned matrix, the method converges slowly. Therefore, sometimes it is worth changing the matrix in a way so it is better conditioned. This can be done by multiplying the equation on both sides:

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{2.43}$$

Now if $M^{-1}A$ has a better condition number than A, it will converge faster, and one can still solve for A once the solution for $M^{-1}A$ is found. A typical variant is the Jacobi preconditioner matrix, which is a diagonal matrix where the diagonal entries are those of matrix A.

The basic algorithm for an Eulerian fluid solver reads:

Algorithm 2: Eulerian fluid solver algorithm

while *in simulation loop* **do**
  at time n
  $\mathbf{w}^A = \text{advect}(\mathbf{u}^n, \Delta t, \mathbf{u}^n)$
  $\mathbf{w}^B = \mathbf{w}^A + \Delta t\mathbf{g}$ //adding external forces
  $\mathbf{u}^{n+1} = \text{project}(\Delta t, \mathbf{w}^B)$
end

Fluid animation covers a wide range of topics and only a few aspects can be sampled in this work. The predictive-corrective incompressible Smoothed Particle Hydrodynamics (PCISPH) method and the conjugate gradient algorithm explained here are examples of the two main approaches for fluid simulation. There are hybrids of these main approaches, like the Fluid-Implicit-Particle (FLIP) concept [20], where the fluid quantities are tracked and advected with

Lagrangian particles but the pressure projection step is performed on an auxiliary Eulerian grid. In the context of this work, the exact type of SPH solver or Euler method is not relevant for the upcoming chapters. This chapter aimed to provide the tools for a basic understanding of the challenging problem of adding small-scale effects to fluid animation.

CHAPTER 3

Distributed VFX Architecture for SPH Simulation

3.1 Brute Force Approach

Section 2.3 shows that the Navier-Stokes equations can also be used to model the small-scale effects like the meniscus effect. No matter if using an Eulerian or Lagrangian method for simulation, the naive approach to add small-scale effects is then to simply reduce the cell or particle size to be small enough to capture the needed detail. But due to the CFL condition, this not only increases the number of needed cells or particles, but also the time steps need to be smaller. This leads to a very high demand for the computation power for the naive approach. This chapter explores the possibilities that moving VFX production to cloud-based infrastructure could offer so that the naive approach still makes sense. The basic idea is to outsource the computation from the artist's workstation to special GPU hardware and thus bring enough computational power to an interactive session so the high resolution needed for small-scale effects is possible.

For many years, leading VFX production houses like ILM or CA Scanline used in-house solutions for physical simulation. A few specialized software vendors such as Next Limit Technologies offer solutions like Realflow [71] that couple to established animation packages. Software packages that started as in-house

solutions have been emerging on the market, such as the fluid simulation software Naiad from Autodesk.

All current solutions, including the above, are either directly integrated into the animation package as extensions like Phoenix [22] from Chaosgroup Software for Autodesk's 3ds Max, where the simulation can be run directly in the base package or the simulation is run in an external application like Realflow and the results are then imported into the animation package as baked simulation data. A good example of a typical pipeline integration in a VFX environment is described by Lagergren [55], whose fluid simulation with a GPU implementation targets the production renderer in the SideFx Houdini system.

Here, a client/server architecture for distributed VFX simulation is proposed that can be seamlessly used within an existing VFX pipeline. Similar to system papers like the one by Parker et al. [74], this chapter describes the design choices that have to be made for the integration of recent research advances into a production pipeline.

The proposed design pattern of this chapter is applicable to many areas in the field of simulation. But fluid simulation is a good use-case because of its high relevance for special effects and since it also affects the rendering aspect of the production pipeline, thus demonstrating how to integrate needed software components in all relevant stages of the pipeline.

A client/server fluid simulation system was designed that allows artists to interactively set up simulations and change parameters inside their familiar animation package while the system executes the numerical calculation for the simulation on a remote system with specialized hardware.

This concept has a variety of advantages. Like in solutions that are directly integrated into the animation packages, the artist does not have to learn new software concepts and only has to explore the feature set of the new client plugin. The artists can stay in their accustomed package. The simulation can run on specialized hardware that can be shared among many artists.

A particle-based method for fluid simulation was chosen because all major 3D packages have built-in particle systems that can handle simulated point data. The coupling to existing particle systems has the advantage that a variety of tools are already available to further use the simulation data, for example, for triggering the creation of splash particles.

An additional contribution is an enhanced direct raytracing technique for isosurfaces that extends the concept of surface particles by introducing blind particles. This reduces the number of particles to be evaluated for rendering and the network bandwidth required for data exchange, which is the main

bottleneck in client/server systems. As a side effect, this also speeds up direct raytracing methods since the rays can skip empty space inside the fluid.

As a case study, the approach was implemented into the 3D package 3ds Max and used external computing capacity based on NVidia GPUs for the server-side. With a user study with VFX professionals, the workflow was evaluated in a production environment.

This chapter uses the PCISPH [85] method for fluid simulation with the kernel functions for density, viscosity, and pressure from Mueller et al. [67] as described in Chapter 2. It employs constant particle size, but adaptive methods could be included as well [4], [104].

For tension forces, the approach by Becker and Teschner [15] is applied using cohesion forces. Integration of the dynamics of the particle system employs explicit first-order Euler integration because it delivers sufficient numerical quality at high computational speed. However, higher-order schemes could be easily used instead, if necessary.

3.2 SYSTEM ARCHITECTURE

The main focus of this work is the practicability of efficient fluid simulation in a VFX environment. To overcome the computational bottleneck of simulating a huge amount of particles on the workstation itself, the computing power is outsourced to an external GPU server. Still, the results of the simulation steps should be visible as soon as possible, and steering the simulation parameters should reside in the 3D scene in the main package.

To achieve this level of interactivity, a client/server architecture is used; see Fig. 3.1. The client resides in the commercial 3D package and sends simulation parameters to the server while requesting simulation results for specific frames. Decoupling of simulation computation and scene management has the benefit that several users can share specialized hardware. GPU blades like the Tesla workstations became affordable for smaller studios. Outsourcing simulation work to these specialized servers is a logical consequence.

The server runs on external hardware and identifies the user and scene. The user can change parameters for the scene in the 3D package and then start a simulation for a specific frame range. The server will run the simulation in a background thread and store the results in a local disk cache. In the host application, the simulation is an operator in the particle simulation framework. The operator holds the simulation parameters and the current state of the particles as input parameters for the simulation and receives the particles over the network. The SPH-related forces are computed in the simulation server and allow the host application to add forces from the integrated particle system

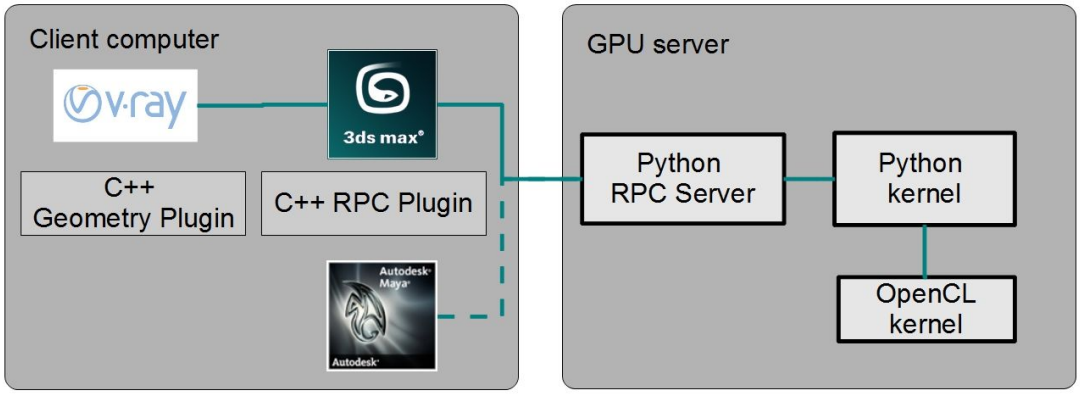


Figure 3.1: Architecture overview: outsourcing simulation tasks to remote hosts.

Figure 3.2: SPH simulation coupled with an IK skeleton. Background image with friendly permission for this usage from Andreas Nilsson [73]. A video sequence of the animation can be found in the video appendix [63].

and then do one integration step for all forces. This opens the way for a variety of effects. With this seamless integration, for example, the interaction of the SPH simulation with forces coming from an inverse kinematics (IK) skeleton that is driven with motion capture data is possible as shown in Fig. 3.2.

In the implementation, the client is integrated into 3ds Max. However, other host systems such as Autodesk Maya or SideFx Houdini would work equally well. For maximum flexibility, the client is integrated into the Particle Flow particle system as an operator using the Particle Flow software development kit (SDK).

The communication between the client and the server is implemented via remote procedure calls (RPCs). The main problem in implementing a remote SPH fluid simulation is the huge amount of data to be transferred. A typical SPH-VFX shot uses several million SPH particles. The simulation data has to be stored at least for every frame. If the host system needs subframe precision to calculate secondary effects like, e.g., additional spray particles, the data has to be saved even more frequently. This computed data has to be stored and streamed over the network. Hence, hard disk space and bandwidth rapidly become critical factors and need careful decision-making regarding file format and message protocol.

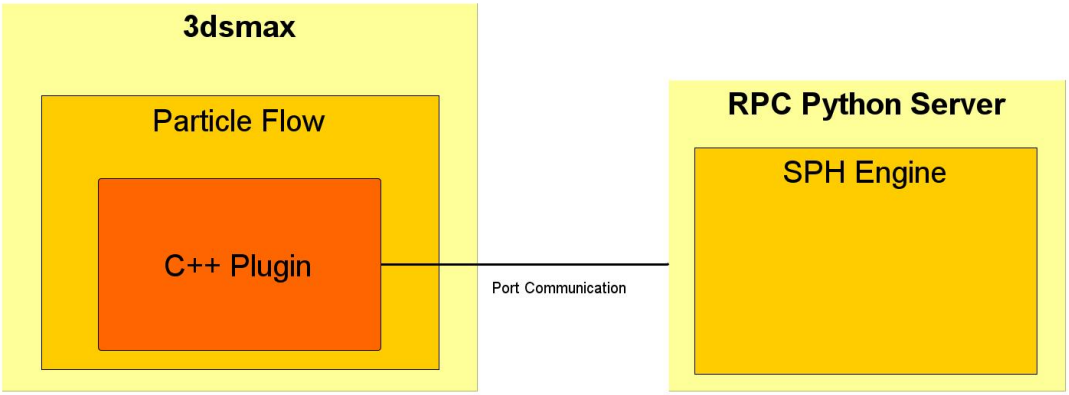


Figure 3.3: Remote procedure call architecture.

As network message protocol, the msgpack [78] protocol is a good choice, which is a bandwidth-efficient protocol for binary data for which implementations in C++ and Python exist. In addition, it allows compressed communication with little overhead. The integration with 3ds Max was done using its event-driven particle system Particle Flow. Particle Flow can display all particle emitters and operators in a node-based editor visualizing a graph of nodes. Several data channels can flow through this graph. With the Particle Flow SDK , new channels and operators can be added to the system using C++. Fig. 3.3 illustrates the remote procedure call architecture.

For this implementation, a new operator is added that holds the msgpack-rpc C++ client. This operator then sends remote procedure calls to the server. The client can request the particle count for a specific time. The system exchanges position, velocity, and color for each particle. The color value will be written to the color channel in 3ds Max. It is used to send the particle type and particle density. Float32 values are used for each array element. This leads to memory requirements of $(9 \times 4) = 36$ bytes per particle, which comes to about 3.6 Megabyte (MB) per 100 k particles. This can be compressed to about 60 percent using zip compression. The operator unpacks the data and then sets both the particle count of the current Particle Flow graph and the positions and information about the particle type.

All performance-critical modules of the simulation were written as parallel code in Open Computing Language (OpenCL) to run on both multi-CPU and GPU setups.

3.3 SIMULATION

The SPH simulation is embedded in the existing particle system of the host application 3ds Max called Particle Flow. The Particle Flow framework em-

ploys an event-driven model, based on the concept of events and operators. Operators describe and modify particle properties such as speed and direction over a period of time. Individual operators can be combined into groups called events. Each operator provides a set of parameters that define particle behaviors during the event. Particle Flow continuously evaluates each operator and updates the particle system accordingly. Particles can be sent from event to event using tests. Tests let you connect events in series. A test can check for certain properties of the particles. Particles that pass the test move on to the next event, while those that do not meet the test criteria remain in the current event [12]. The simulation code is implemented as an operator of Particle Flow. This decouples the SPH simulation from the computation of other forces that may act upon the particles.

Particle Flow has its own integration routine. To avoid 'double' integration a Particle Flow node can overwrite the default integrator with a custom implementation. This enables the integration of the particles in the operator and makes sure that boundary conditions and collisions are handled properly. By reading out the acceleration channel, the Particle Flow forces can affect the simulation. The acceleration due to the Particle Flow forces changes the velocity on the client side while the SPH acceleration is added on the server. The mass is defined as constant for all particles; therefore, the accelerations can simply be added:

$$\mathbf{a}_{total} = \mathbf{a}_{SPH} + \mathbf{a}_{ParticleFlow}. \tag{3.1}$$

The PCISPH technique [85] as described in Chapter 2 was adopted with strategies for GPU implementation according to Goswami et al. [34]. PCISPH offers a good time/visual quality ratio, high robustness, and easy implementation. However, the choice of simulation algorithm should not affect the system architecture. One of the benefits of this approach is that maintaining and changing the simulation implementation is decoupled and transparent: it can be done without changes to the local software setup of the client workstations as long as the parameter set stays the same.

3.3.1 Neighbor Search

SPH simulation is well suited for a parallel implementation since the particles can easily be distributed to the computing units. Efficient neighbor search for each particle is the challenging part for parallel implementations. The decision for an acceleration structure was based on the requirement that a memory-efficient structure that can run on the GPU is necessary. Also, a concept that can be extended to a multi-GPU scenario is preferable, similar to the multi-GPU work by Valdez-Balderas et al. [98]. The system presented here

is based on the approach from Goswami et al. [34] using a space-filling curve as described in Chapter 2.

3.3.2 Collision Detection

Collision geometry is sent to the server with an RPC call by sending the polygon positions and a global transform matrix of the object for each frame. A hash value is generated for the polygons of each object based on the vertex positions. The polygons are cached on the simulation server and are only updated if the hash value changes. To handle collisions, the particle trajectories are intersected with the triangles in both the PCISPH prediction step and the final integration step. If a collision is found, the position of the particle is corrected to the intersection point and its velocity is set to zero. The additional pressure correction iterations of the PCISPH method propagate the pressure back into the fluid and lead to visually satisfying results, as experience showed. Therefore, more sophisticated boundary handling methods were not needed for the use cases described in this chapter.

3.3.3 Blind Particles

To reduce the amount of data required for rendering, the author introduces the use of blind particles. Only particles near the surface contribute to the surface generation and the rendering process. The particles inside the volume are not of any interest for visualization. This observation was used before to accelerate raytracing. We also exploit this fact for optimized disk caching and reduction of bandwidth requirements. Recent works [34], [109], [7] identify the surface particles from the simulation and only use those for surface generation. Akinci et al. [7] use surface particles for surface generation in a tesselation process based on Marching Cubes, but not for directly raytracing the surface. They store the surface information on the grid that they use for their Marching Cubes algorithm while the method described here tags the underlying particles with the particle type property directly.

However, identifying only surface particles is not sufficient when used with a production renderer. Fig. 3.4 (a) illustrates this problem. In this example, the inner red surface only emerges because the inner particles were deleted. This problem arises for commercial raytracers because the core components of the raytracing algorithm cannot be changed by plugins. Therefore, one cannot decide if the surface is valid or not by the algorithm, especially if camera rays that start inside the fluid should be possible. A ray that enters a fluid and passes the surface particles will generate a surface on the inside of the boundary particle hull. For the ray, it could also be a small splash particle or a thin fluid stream.

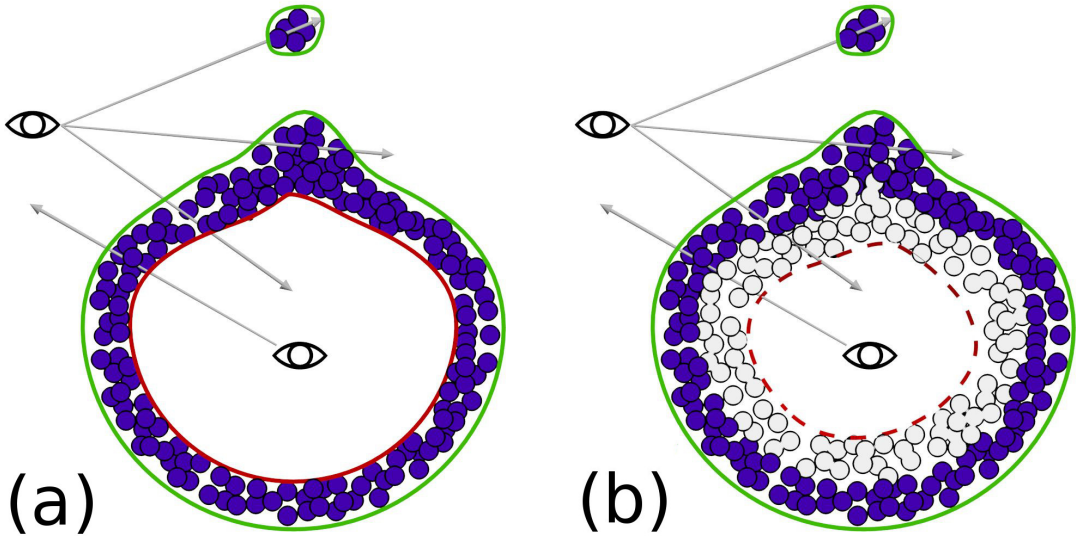


Figure 3.4: Surface generation without and with blind particles. The green line depicts the outer surface, the red line the inner surface. Rays are shot from different positions towards the scene. On the left, the raytracing algorithm will create the inner surface as if the drop was hollow. On the right, the blind particles will prevent surface generation and the drop renders as if it was completely filled with water.

To solve this problem, I introduced blind particles. This method does not only identify the surface particles but also a thin layer of particles tagged as 'blind' around them; see Fig. 3.4 (b). Blind particles prevent the generation of the inner surface. The rest of the particles is only relevant during simulation. Omitting particles would normally create new surfaces inside the fluid where they are missing. The blind particles will tell the surface generation algorithm to not generate a surface where their field value would normally create an isosurface. For identifying the surface particles, the method described by Goswami et al. [34] is adopted. A blind particle is tagged as blind if at least one neighbor in a threshold distance was tagged as a boundary particle. Fig. 3.5 shows a typical scenario with blind particles, surface particles, and particles that are to be omitted.

3.3.4 Implementation Details

Similar to Goswami et al. [34], the zCurve approach is used for parallel implementation on the GPU. The method temporarily stores the neighborhood of each particle in memory for each timestep to reuse the information in the prediction/correction loop.

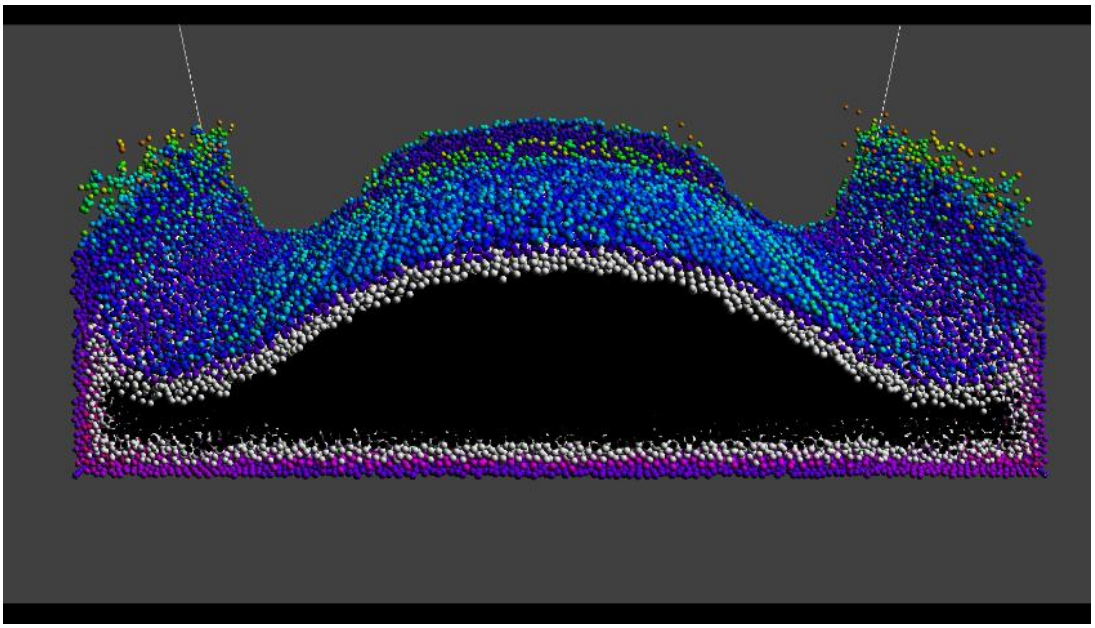


Figure 3.5: SPH particles color-coded by particle type. Black: particles to be omitted, white: blind particles, colored: surface particles color-coded by pressure.

For the simulation code, OpenCL over CUDA was chosen because of the ability to run it efficiently on CPUs as well. For the host code, the Python OpenCL integration [52] was chosen. Experience showed that kernels run at the same speed independent of whether host code is written in C++ or Python. For array operations that have to be executed on the host, the NumPy library is used that offers highly efficient algorithms.

The advantage of a Python/OpenCL-based framework is good portability. The code ran on all combinations of Windows/Linux and GPU/CPU with only little modifications. However, some optimizations that would be possible for GPU-only code were sacrificed for CPU compatibility. For example, the use of shared local memory was avoided in the kernel code.

3.4 RENDERING

My contribution to the rendering of fluid surfaces is the integration of current fluid rendering algorithms in a VFX pipeline. Often, rendering is a step separate from animation and simulation. To adapt to this separation, the rendering routines were designed as a geometry plugin for 3ds Max from Autodesk and the raytracer VRay from Chaosgroup. This renderer is available for all major 3D packages and is widely used in VFX companies in Europe.

In general, two ways are used to render fluid surfaces. One approach is to generate a triangle mesh that is then rendered with the standard triangle pipeline. Akinci et al. [7] proposed an efficient parallel implementation for the surface reconstruction. The other approach is direct raytracing of the surface where the intersection with the surface is found for every ray [39]. Fraedrich et al. [29] accelerated direct rendering of fluid surfaces on the GPU by sampling the particle values to a perspective grid. A surface reconstruction algorithm and a direct raytracing routine were implemented and both solutions were compared.

In the preparation phase, a routine fills the particles into a bounding volume hierarchy (BVH), taking the radius of each particle into account. The implementation presented here differs from other implementations [35] regarding that if motion blur is required, this implementation uses both the position at the beginning of the motion blur interval and the end to define the leaf nodes. Each particle carries velocity information; the ray request from the raytracing system has time information. With this information, each sample can calculate the exact positions of the particles in the intersected BVH leaves that were generated for the full-frame length. Fig. 3.6 shows how the BVH is used for rendering motion-blurred isosurfaces.

As a first step, a routine finds the intersected leaf nodes and then sorts those nodes by their intersection point along the ray. Then, it marches along the ray from the first intersection point into bounding boxes and evaluates a level set function for the particles in the leaf node. Both the simple Blinn blob and the surface function according to Zhu and Bridson [112] was implemented. If the sign of the result changes from one evaluation to the next, then the surface lies between those marching steps. The accurate position is found via binary search.

The routine omits the intersection if most particles involved in the calculation of the level set are flagged as blind particles. This robustly handles all secondary rays like shadow rays, reflection/refraction rays, and rays for indirect illumination. Also, rays that start inside the fluid are handled correctly. Fig. 3.7 compares rendering with and without blind particles.

Direct raytracing is very fast for opaque materials. For ray marching near the boundaries, only outer leaf nodes of the BVH have to be considered and only a few particles add to the level set function. Once the first intersection is found, the function is finished and may return. The slightly longer raytracing times compared to the intersection with polygons are compensated with the much faster preparation times per frame, since no marching cubes [60] or similar meshing phases are needed for explicit isosurface extraction. Especially in cases where huge water masses are simulated, but the camera captures just a fraction of them, the direct rendering method can speed up rendering

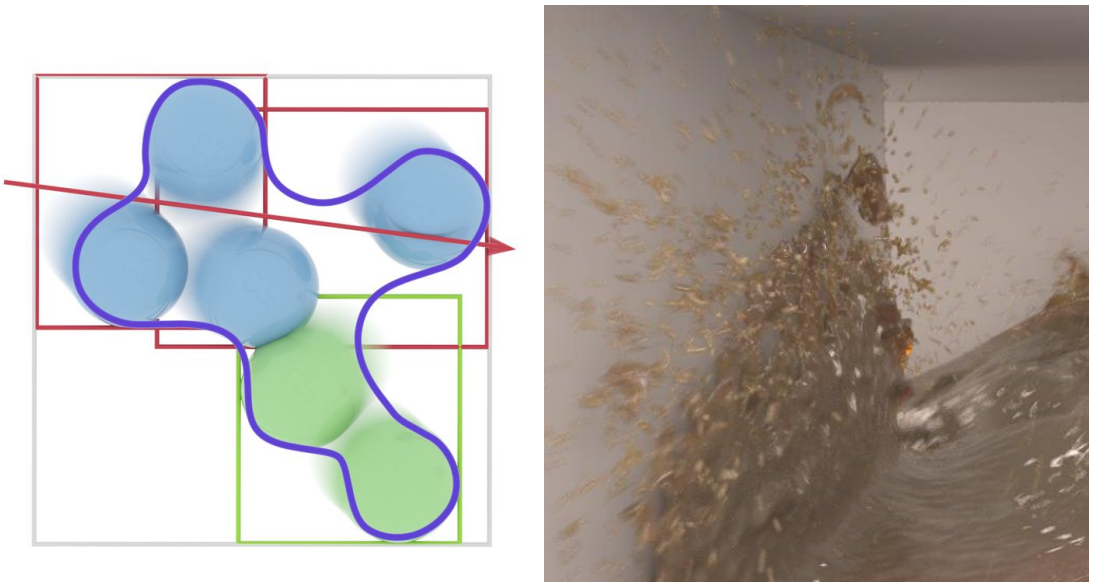


Figure 3.6: On the left you see a BVH for a motion-blurred isosurface (blue line) that is constructed from moving particles. Both the start position and the end position are taken into account when creating the bounding boxes. The ray (red arrow) intersects the red boxes and the particles are used for the calculation. The green box is not intersected and its particles are omitted. On the right is a rendering of a moving fluid that uses this BVH method for direct raytracing of an isosurface with motion blur.

substantially. An important advantage of the direct raytracing approach is the handling of 3D motion blur. Mesh-based rendering methods have to calculate multiple meshes per frame to obtain clean motion blur in order to compensate for changes in the mesh topology. Modern raytracers can render motion blur based on velocity information per vertex, but this approach cannot consider topology changes, e.g., merging of fluid drops. For direct raytracing, it does not matter if the topology changes inside the time frame of the motion blur. In Fig. 3.6, the isosurface for a single point in time is drawn as seen by a specific ray sample. For each sample, the algorithm calculates the exact position of the surfaces at the requested point in time.

However, the advantage of faster rendering times is lost for transparent materials like water. Here, ray marching has to continue through the material. Also, reflections inside the fluid added to the rendering times. Here, the caching strategies VRay provides for polygons outperform direct raytracing. To allow for polygon-based raytracing, a multi-threaded marching cubes algorithm to create polygons was implemented.

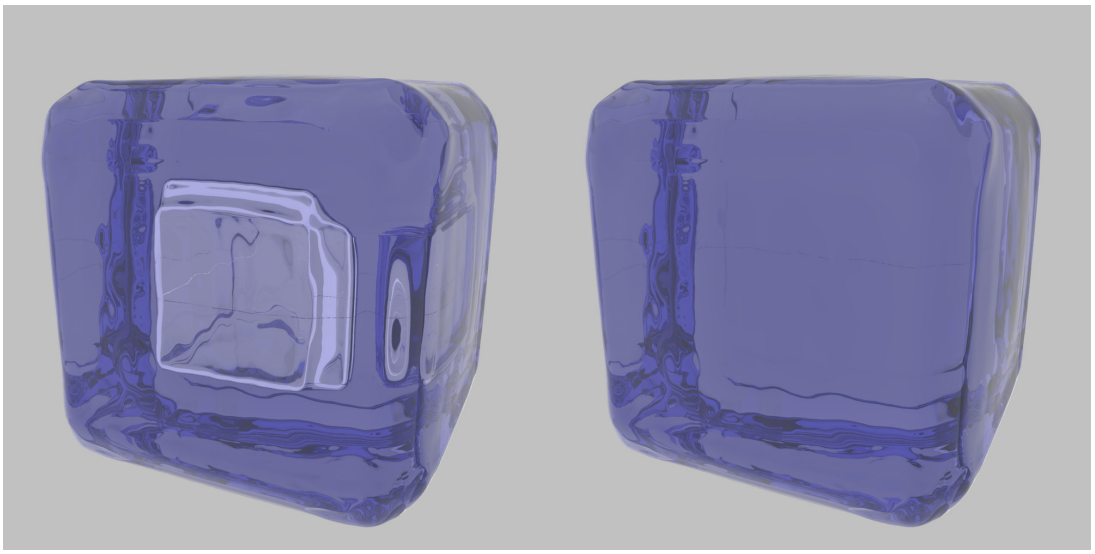


Figure 3.7: Rendering of a particle cube where inner particles were omitted: (left) without blind particles and (right) with blind particles.

Table 3.1: Simulation time for 100 frames.

Scene Stepsize Δt	Dambreak 20k 0.01s	Dambreak 125k 0.01s	Dambreak 1.25M 0.01s
Local Intel i7	0h:01m:27s	0h:13m:29s	2h:54m:08s
Local NVidia GTX 570M	0h:00m:48s	0h:05m:06s	1h:09m:28s
Local NVidia GTX 680	0h:00m:19s	0h:02m:16s	0h:29m:34s
Server/client GTX 680	0h:00m:20s	0h:02m:19s	0h:30m:17s

3.5 RESULTS

To validate the approach, on the one hand the raw performance was tested and on the other hand a user study conducted to check the practicability of the system.

3.5.1 Performance Results

Performance numbers are provided for the simulation and the rendering components of the system. All tests use a dam-break simulation designed by the author. Fig. 3.8 illustrates different time steps of the simulation; also see the accompanying video in the video appendix [63]. Different volume quantities

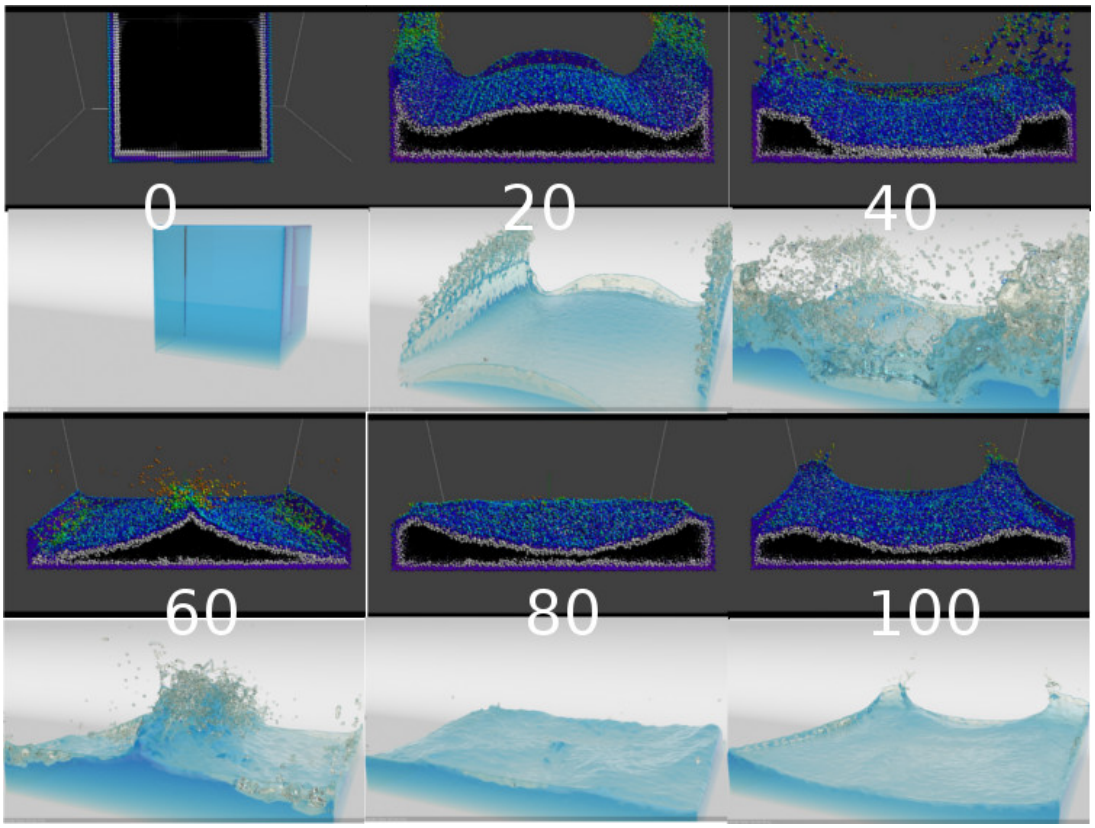


Figure 3.8: Different time steps of the dambreak simulation.

were used for the SPH simulation keeping the kernel radius at 4 cm, and time step at 0.01 s (with 20 k, 125 k, or 1.25 M particles). The test environment was a mobile workstation equipped with an NVidia GTX 570 M GPU and Intel I7 CPU (2.0 GHz) and a desktop workstation with an NVidia GTX 680. In the client/server configuration, the mobile workstation was the client with 3ds Max and the desktop workstation was the simulation server. The network was a 1 GBit Ethernet connection.

Table 3.1 documents the simulation times for the different hardware platforms and SPH quantities. The simulation times include the transfer of the particle data between client and server for each frame and, therefore, may be slower than times presented in other SPH real-time papers. However, the difference between running the server on the same machine ("local" in Table 3.1) and running over the network was only about 2 percent. The advantage of having access from the mobile workstation to the computing power of the GTX 680 card in the desktop workstation, which resulted in about twice the speed, by far compensated the communication overhead.

Especially, time-critical productions may benefit from the flexible use of external compute resources.

Table 3.2: Rendering times (min:sec).

Scene		Dambreak 20 k	Dambreak 200 k
Marching Cubes	opaque	0:36	2:46
	transp.	0:39	2:27
Direct raytracing	opaque	0:18	2:01
	transp.	0:55	6:18
Direct raytracing blind	opaque	0:18	1:57
	transp.	0:55	4:43

Table 3.2 shows the rendering times on the mobile workstation for the small and medium-sized dam-break simulations. The rendering times were recorded for frame 40 of the simulation, where a large number of active particles are present. As shown in Fig. 3.10, there are other time steps of the simulation that have much fewer active particles. In those cases, savings from direct raytracing are even more pronounced than for frame 40 of the animation.

Finally, Fig. 3.11 compares rendering times between the blind particle method and the conventional method. The direct raytracing implementation is especially helpful in the setup phase for low-resolution test rendering. Especially for small cropped render tests without antialiasing, direct raytracing outperforms methods that need preparation steps with meshing. This allows fast iterations in the lighting phase.

Longer preparation times for polygon generation pay off in more complex scenes with multiple ray bounces. For each ray sample, the scalar field of the isosurface has to be evaluated multiple times for the binary search in order to find the intersection point. This is a costly operation. For small resolutions without antialiasing and only single ray bounces, this is still faster than surface reconstruction. Antialiasing schemes dramatically increase the rendering time for direct raytracing while only moderately increase the rendering time for polygons. This can be seen in Table 3.2 for frame 40 of our dam-break simulation. All tests were performed with an image resolution of 1280×720 pixel and fixed-rate antialiasing. The conclusion is that direct raytracing should be used in the setup phase, when small resolution test renders are made, or in situations where only portions of the simulation volume are seen by the camera.

The presented geometry plugin supports all VRay render elements. For the final rendering, the plugin can produce additional passes that could be used in a compositing step; see Fig. 3.9. The passes are rendered on the fly with the beauty rendering at almost no additional rendering time cost and saved to channels in the OpenExr file. Being able to deliver all requested render

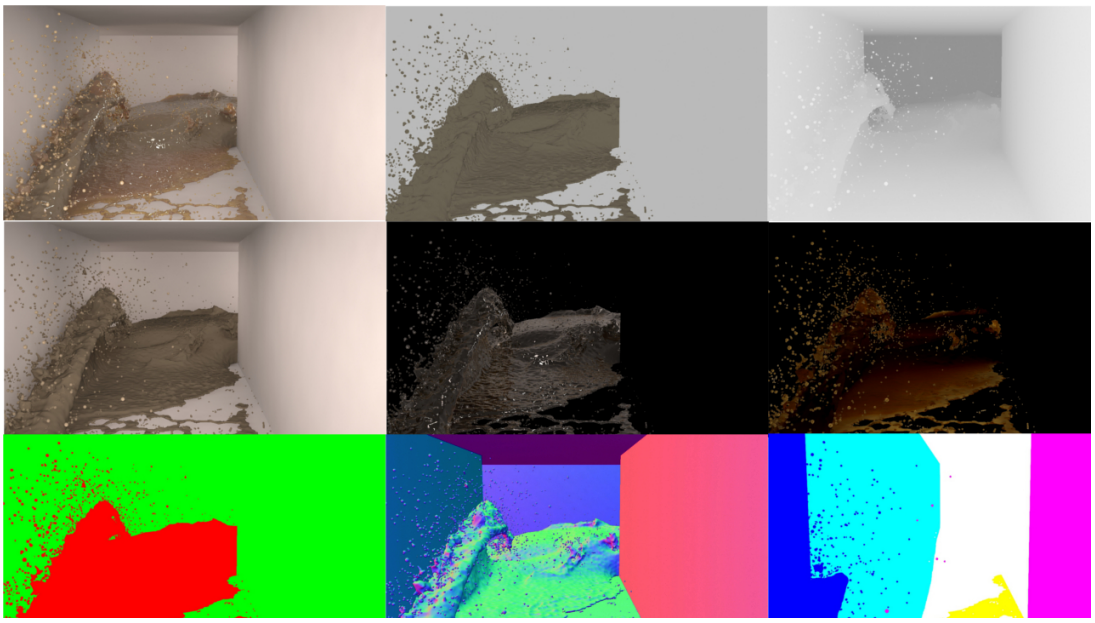


Figure 3.9: Different render element channels: beauty, diffuse, z-depth, global illumination, reflection, refraction, mask, normal, and world coordinates.

element passes is an important factor for successful integration in today's production pipelines.

3.5.2 User Study

The usability and effectiveness of the system were assessed by conducting a user study with VFX professionals. The study is mainly of qualitative nature where the participants were commenting on their usage of the system and filling in a survey afterward. In addition to gathering information to improve the usability of the system (as part of a formative process), the goal was to test two hypotheses: "SPH solver integration into Particle Flow improves the workflow in 3ds Max" (Hypothesis 1) and "Direct raytracing is suitable for VFX production" (Hypothesis 2). The detailed results of the study can be found in Appendix A.

The study was designed to test realistic work environments and tasks. Therefore, specialists in the field of fluid-related VFX were recruited. Due to the highly specialized user group, the number of participants was small (three). Therefore, a qualitative user study design was chosen: The think-aloud method was used [57] in addition to questionnaires to obtain maximum feedback from each individual. According to Nielsen [72], three expert users can be sufficient for a think-aloud study. A screen and audio recording were later transcribed into text and selected comments were then categorized into the phases 'initial

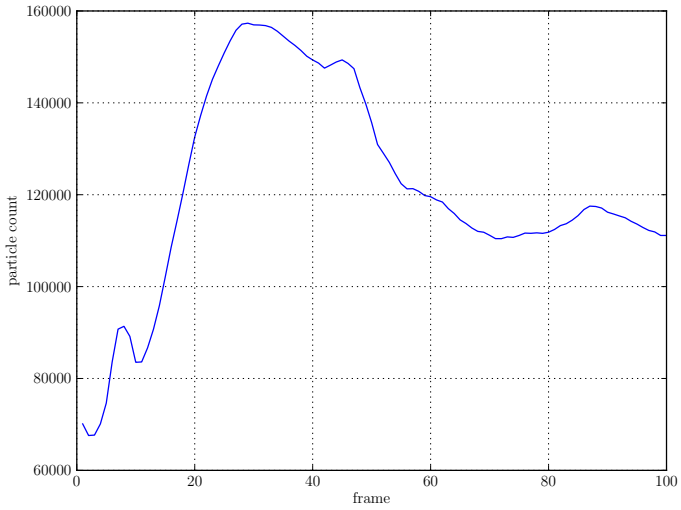


Figure 3.10: Particle count of active particles (boundary and blind particles) of the 200 k dam-break simulation.

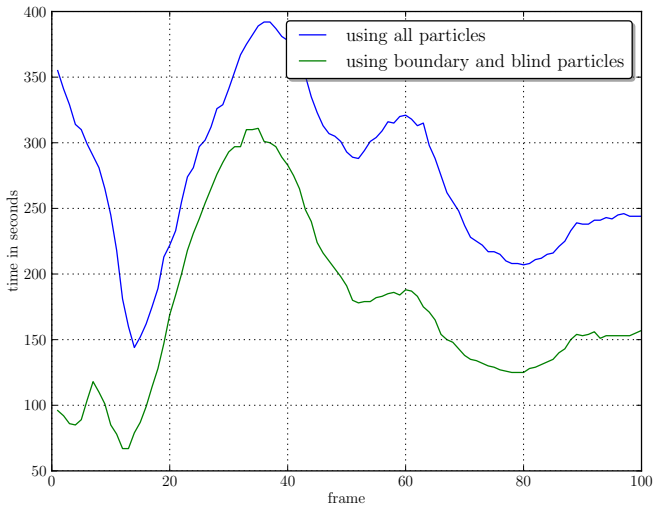


Figure 3.11: Rendering times for the blind particle and conventional methods for the 200 k dam-break simulation.

setup', 'tweaking simulation', 'tweaking surfacing', 'tweaking render settings', and placed in a review document. First, the experience level of the users was identified with a background questionnaire. Then, the experience level was determined by the amount of time per week the participants work with certain software packages or areas of the VFX pipeline and the years of experience they have in their field.

All participants had more than 3 years of professional experience with 3D software and spent more than 6 hours per day working with 3ds Max. All participants were working on fluid-related VFX projects in their jobs at the time of the user study. Since VFX professionals with a fluid background and 3ds Max experience are scattered over the world, the user study was performed remotely with screen sharing sessions. The participants temporarily installed the plugin on their workstations. In a remote session, the author guided the participants through the installation and gave a brief introduction with an online presentation of the system, and provided sample scenes for 3ds Max. After an introductory session of about 30 minutes, they were asked to design a dam-break-like animation. The task was easily explained and can be solved in the short amount of time available. As an additional requirement, they were asked to add forces from 3ds Max to the simulation like, e.g., a vortex to explore the coupling with 3ds Max forces. The participants were instructed to think aloud while they were working to protocol their first impressions.

After the test, the participants filled in an online survey with questions about the usability of the system. Included in these were questions about different areas of the system like *"Integrating fluid solver in Particle Flow allows me to achieve a greater variety of effects than a standalone solver"* and the 10 standard System Usability Scale (SUS) questions [21]. The duration of the test was between 30 and 45 minutes.

All participants agreed that the overall workflow is better if the fluid simulation is integrated into the 3D content creation software in contrast to standalone fluid simulation applications. The usability was also rated very good in the SUS questionnaire. This is attributable to the fact that the solver blended into the interface for which they were experts. In particular, all users appreciated the flexibility that the integration into Particle Flow offered.

The participants were also asked about their subjective opinion on the visual quality of direct raytracing of fluid blobs in contrast to meshing approaches. All agreed that direct raytracing offers superior image quality; no participant attributed it as slow. All would consider it for their next project. The detailed questions and results of the questionnaires can be found in Appendix A.

The participants were also asked for unstructured feedback. Some of the representative feedback includes: "The integration of an SPH solver into a particle system is basically interesting. It often happens in daily work that you have to add small fluid simulations on top of existing particle simulations, where you don't want to setup a big system. For example liquid spurts in battle scenes, where a complete fluid simulation would be too much, but still small effects are needed." Or: "This should actually in theory kill the performance because we are raytracing into an isosurface and that is something you shouldn't do. Sure enough it is going slightly slower but testament to the

quality of the mesh it's intersecting – even at the low settings – it is really not." Or: "It is really a dream to be able actually stop a render this quickly and go back to your settings, and change them, tweak them, press f9 to re-render and it is there.... Seriously, on a daily basis I have to wait 15 minutes between stopping a render and releasing all memory possible from the computer it takes minutes and then I make the one small change like 'I need two more of this' and press f9 and wait 10 minutes before the fist bucket is on screen."

In summary, the user study provides a preliminary indication that this approach provides a useful integration of SPH simulation in the VFX workflow (Hypothesis 1) and that direct raytracing can be suitable for certain aspects of VFX production (Hypothesis 2).

3.6 SUMMARY

This chapter investigated Research Question 1 about strategies to increase the efficiency of existing simulation methods, so the resolution can be increased. The presented approach shows that external and distributed computing resources can be integrated into the established 3D workflow of VFX production companies seamlessly using commercial software packages allowing interactive sessions. The user study confirmed the good utility of the approach for domain experts. The cloud computing paradigm and flexible offers from cloud computing vendors open the door for smaller animation studios to run complex simulations for 3D production work on high-end GPU clusters without having to invest in specialized hardware. The approach reduces the memory footprint of particle caches without any visual difference. This is especially important if the data has to be transferred over the network. The idea of blind particles is independent of the simulation concept used and can be easily integrated into existing pipelines with savings in both rendering time and storage requirements. The performance tests confirmed that the distributed simulation leads to negligible communication overhead.

Although the proposed measures counteract, to some extent, the unfavorable nature of the cubic increase in render time with increased resolution, the additionally needed reduction of the time step due to the CFL condition inevitably brings the naive approach to its limits. The consequence is that for adding small-scale effects to simulations, simply increasing the resolution is only possible to a certain degree. Increasing the resolution of a whole simulation system to capture one particular small effect at the surface wastes computation time in areas where small-scale effects are not expected. The next chapter will specialize on the meniscus effect and use the knowledge about its nature to spend computation time where it is needed to create this particular effect without simply increasing the resolution of the simulation.

CHAPTER 4

Direct Raytracing of a Closed Form Meniscus

Chapter 3 investigated the naive approach to simply increase the resolution of simulations for adding small-scale effects. Although it is possible to increase the resolution to a certain extent, this brute force approach is always limited since with increasing resolution not only particle counts rise, but also time steps need to get smaller. Distributing the work can help here, but the second research question aims for smarter solutions.

4.1 Introduction

In the last two decades, fluid simulation for visual effects has seen great advances in the domain of free surface motion. However, most work has been done for large to mid-scale simulations, where the capillary effects at the interface to solid boundaries are not a prominent feature. Usually, for simulation purposes, the surface interfaces to solid objects are treated as impermeable boundary conditions. In most cases, surface tension between a liquid and a solid is ignored. In large-scale simulations, the forces of surface tension are negligible. Putting computational work into their calculation is therefore not reasonable.

This is quite different for synthesizing small-scale liquid motion, such as for water droplets on a surface. Surface tension forces become strong compared to gravitational forces. Here, one wants to avoid the computational overkill

that comes with small-scale simulations. As will be presented, surface tension effects can be achieved at render time without any extra effort for the numerical simulations. Previous work requires either heavy computation (for high-resolution simulation) or high-resolution meshes to model the meniscus.

One visually interesting phenomenon of surface tension forces is the effect of the fluid meniscus at the contact line between fluid and solid. The meniscus is a curved shape at the border that ensures that the fluid is approaching the solid at a certain characteristic contact angle. These small curved areas gather highlights from many more directions than a flat fluid surface. This creates an observable visual difference because of the produced highlights even in larger-scale settings where the geometry of the meniscus itself might be hardly visible.

This chapter introduces a new approach that uses a closed-form of the meniscus shape of the fluid interface in order to add a small-scale curved surface along the contact line, to mimic a fluid simulation at render time. The method generates the surface locally at the ray–surface intersection step of the raytracing algorithm without generating triangles. The solution operates on the isosurface of a fluid simulation and the distance fields of the collision objects and is therefore applicable to a wide range of simulation methods.

The research showed that the method puts only little computational overhead to the ray intersection step and still generates accurate contact angles and plausible meniscus shapes. The decoupling of large-scale simulation and small-scale meniscus effects improves the overall workflow since the large-scale simulation can be cached and the meniscus effect can be fine-tuned for rendering.

4.2 PREVIOUS WORK

Already in 1976 Saville simulated a contact angle with a molecular dynamics simulation [79]. As early as in 1993, Kaneda et al. [49] described a method to render water droplets on a glass plate considering interface dynamics and the contact angle between water and solid. They use simple environment mapping and a fixed spherical shape for the drops. Fournier et al. [28] simulate the flow of droplets on arbitrary meshes with particles. They also use a fixed shape for the droplets. Kaneda et al. [48] improve the simulation of drops running down a surface with a probabilistic method for merging and pathfinding for the water-based on affinity and dry conditions. These methods with fixed shapes are limited to individual, separated drops and cannot merge or split the fluid surface.

Yu et al. [107] propose the use of metaballs for a more flexible model of the drop shapes. They change the shape of the droplets depending on the gravitational

field and merge drops within a certain distance. They use position-dependent surface properties to achieve irregular drop shapes. In contrast to our method, they do not consider material-dependent contact angles and do not use any physical simulation.

Zhao et al. [111] fully simulate droplet behavior on level set grids. This approach produces physically correct animations for drops but needs extensive computations, even for a single drop. The authors do not consider solid collision surfaces. Wang et al. [99] present an algorithm to fully simulate water drops running on solid surfaces with level set grids. They take into account contact angles at solid–fluid interfaces. The methods by Zhao et al. and Wang et al. include the capillary effects in the physical simulation and perform a regular surfacing step at the end. Since the resolution of the physical simulation has to be fine enough to capture the capillary effects at the borders, the approach comes with very high computational costs.

Rendering water drops in real-time applications can be based on height fields [47], [58], [90] or particles in 2D texture space [70], [94]. These methods limit the fluid surface to a narrow band around the collision object and therefore are restricted to specific use cases.

Recent methods use mean curvature flow operators for triangle meshes to produce surface tension effects. Zhang et al. [110] achieve real-time frame rates with physically plausible behavior for small drops. Clausen et al. [23] use Lagrangian meshes to simulate the behavior of fluid droplets with high accuracy. They employ the Lagrangian mesh representation of the fluid surface for a full simulation of the fluid and consider physical effects like viscosity, incompressibility, and tension forces. Our method does not operate on the surface mesh, it neither uses nor influences the underlying physics simulation and can therefore not be directly compared with the mean curvature flow methods. In contrast to the simulation with Lagrangian meshes, our method changes the shape at render time and operates on the scalar field of the isosurface of the fluid. Another mesh-based method is the correction applied for fluid–cloth coupling by Huber et al. [41]. However, they only avoid penetration but ignore the contact angle.

The most related work is by Bourque et al. [18]. They also use a closed-form meniscus solution to displace the vertices of a mesh to form the correct meniscus. In their method, the vertices are displaced independently. The resulting mesh may therefore have self-intersections. This can produce artifacts in certain situations where the fluid surface has frequencies that are in the scale of the meniscus length, or when two solids are immersed in a liquid very close to each other. Their method requires a fine-resolution mesh for the surface correction. Although the simulation can be coarse, the resulting mesh must be fine enough to produce the menisci.

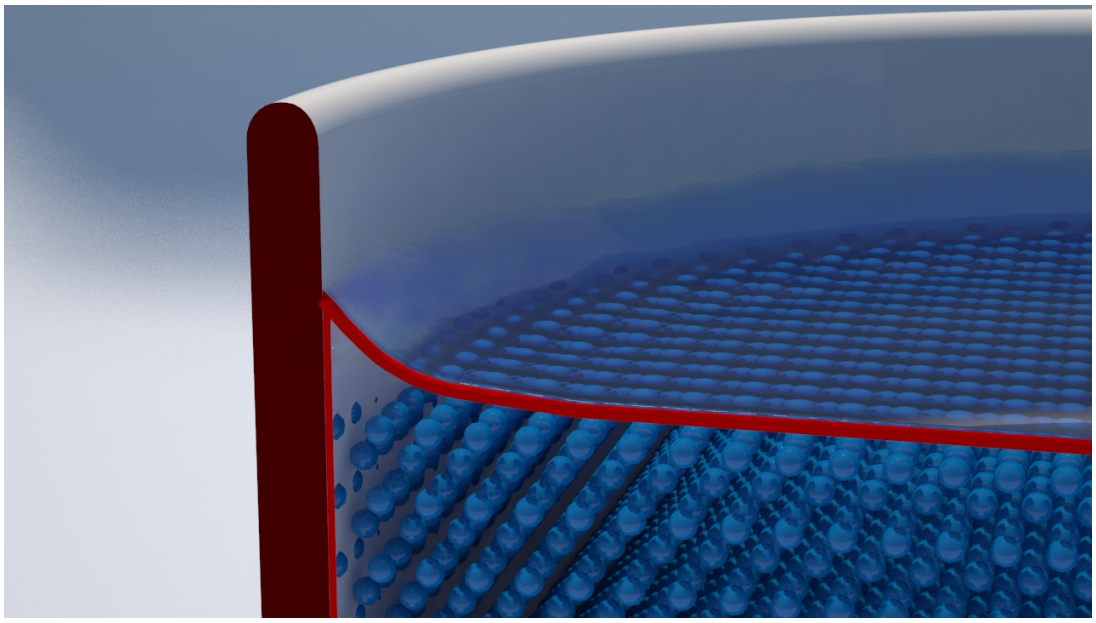


Figure 4.1: Meniscus at vertical wall.

The work of this chapter is also related to the domain of displacement mapping techniques [92]. To address the problem of the high mesh resolution, the method integrates the analytic meniscus solution into the implicit description of the fluid surface. In this way, direct raytracing of the modified, correct fluid surface is possible. In general, direct raytracing of implicit fluid surfaces without using a mesh representation has been extensively researched both on CPU and GPU (e.g., [39], [29], [35], [84]). But none of the published papers considers the menisci of collision objects for free boundary creation. Gourmel et al. [36] combine implicit objects by designing new implicit functions using the gradient functions of the source objects to achieve certain effects at the interfaces, but they do not target the special use case of fluid surfaces.

4.3 THEORETICAL BACKGROUND

The contact angle is well visible in the case of a horizontal fluid surface in contact with a vertical solid plane as explained in Chapter 2. Here, the fluid forms a characteristic meniscus at the contact line, as shown in Fig. 4.1. For this special case, an analytical solution for the shape can be derived. An in-depth derivation is given by Deserno [25]. Here, the relevant material for our approach is briefly stated. At the vertical wall, the fluid interface will rise to a distance y_0 above the horizontal level to achieve the characteristic wetting angle α, as can be seen in Fig. 4.2. The meniscus starts at the wall with the given contact angle and approaches the ambient interface level far from the

wall.

According to the Young-Laplace law, the hydrostatic excess pressure $\Delta p = (\rho_{\text{liquid}} - \rho_{\text{air}})\, g\, y$ must equal twice the pressure due to the interfacial tension σ and the mean curvature H:

$$\Delta p = 2\sigma H\,.$$

The angle ψ against the horizontal plane can be parametrized as a function of the arc-length s along the profile. The mean curvature is $H = \frac{1}{2} d\psi/ds$, as shown by Spivak [87]. This leads to the differential equations:

$$\dot{\psi} = -\frac{y}{l^2}\,,$$

$$\dot{y} = -\sin\psi\,,$$

with the capillary length l:

$$l = \sqrt{\frac{\sigma}{g(\rho_{\text{liquid}} - \rho_{\text{air}})}}\,.$$

The density of water and air are denoted ρ_{liquid} and ρ_{air}, respectively; g is the gravitational acceleration. For the water–air interface, this is $\sigma \simeq 80\,\text{mN/m}$, which results in $l \simeq 2.8\,\text{mm}$. According to Deserno, there is:

$$y(\psi) = 2l \sin\frac{\psi}{2}\,.$$

At the wall with $\psi(s=0) = \psi_0 = \frac{\pi}{2} - \alpha$, there is the meniscus height:

$$\frac{y_0}{l} = 2\sin\frac{\pi - 2\alpha}{4}\,. \tag{4.1}$$

This finally leads to:

$$\psi(s) = 4\arctan\left[\tan\frac{\psi_0}{4} e^{-s/l}\right]\,,$$

$$\frac{x(s)}{l} = \frac{\frac{s}{l}\cosh\frac{s}{l} + (\frac{s}{l}\cos\frac{\psi_0}{2} - (1-\cos\psi_0))\sinh\frac{s}{l}}{\cosh\frac{s}{l} + \cos\frac{\psi_0}{2}\sinh\frac{s}{l}}\,,$$

$$\frac{y(s)}{l} = \frac{2\sin\frac{\psi_0}{2}}{\cosh\frac{s}{l} + \cos\frac{\psi_0}{2}\sinh\frac{s}{l}}\,.$$

In addition, Deserno proves the translational invariance of the curve in x direction for the starting angle ψ_0. Therefore, one can choose any starting

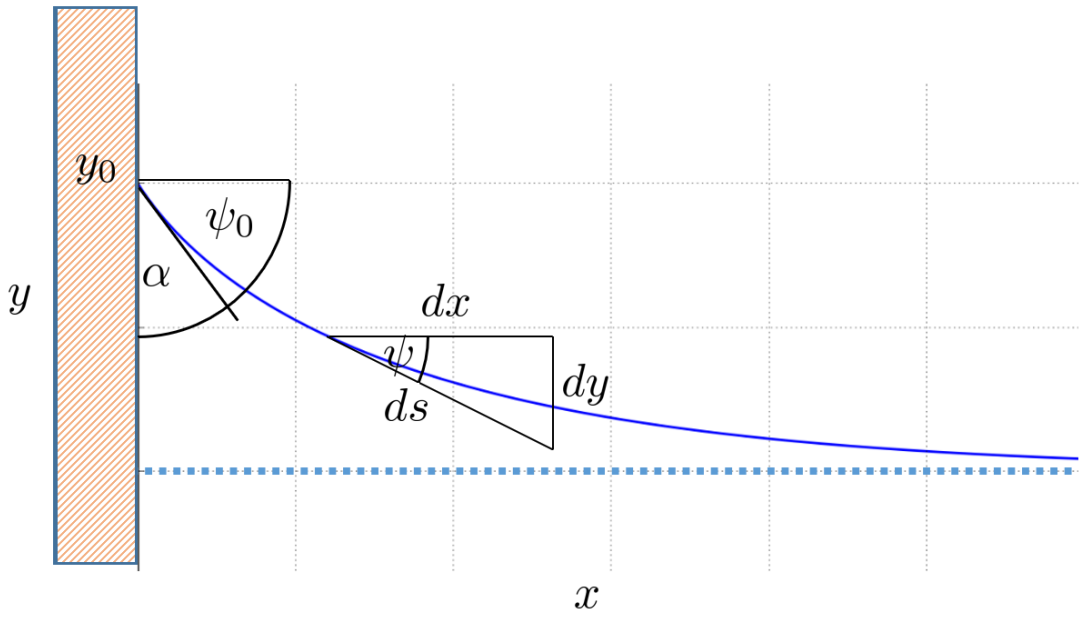


Figure 4.2: Meniscus shape with angle ψ as a function of the arc-length s along the profile.

angle and later shift the curve to the right location to calculate the shape for various contact angles, as can be seen in Fig. 4.3.

According to Yuan and Lee [108], there are more properties than the interface tensions, like geometry and flow rate, that also influence the contact angle. The contact angle is also changing during droplet impact due to the contact angle hysteresis where the advancing contact angle and the receding contact angle differ. Bourque et al. [18] show that using different meniscus functions creates negligible differences in the created caustics. Therefore, it is expected that differences caused by those additional parameters are visually imperceptible and these parameters can be ignored for the solution.

4.4 IMPLICIT MENISCUS MODEL

The above analytical solution of the meniscus height can be used to produce a physically correct meniscus. First, it can be observed that the 1D closed-form solution of the meniscus is solely based on the distance to the collision object. The fluid surface can also be described by a distance function. If the fluid approaches the solid at an angle of 90 degrees and is given a point on the original fluid surface, the fluid's distance function can be corrected by adding the result of the meniscus solution. The meniscus solution can be seen as a height field that displaces the original surface. This way, the meniscus solution can be used to modify the implicit surface function.

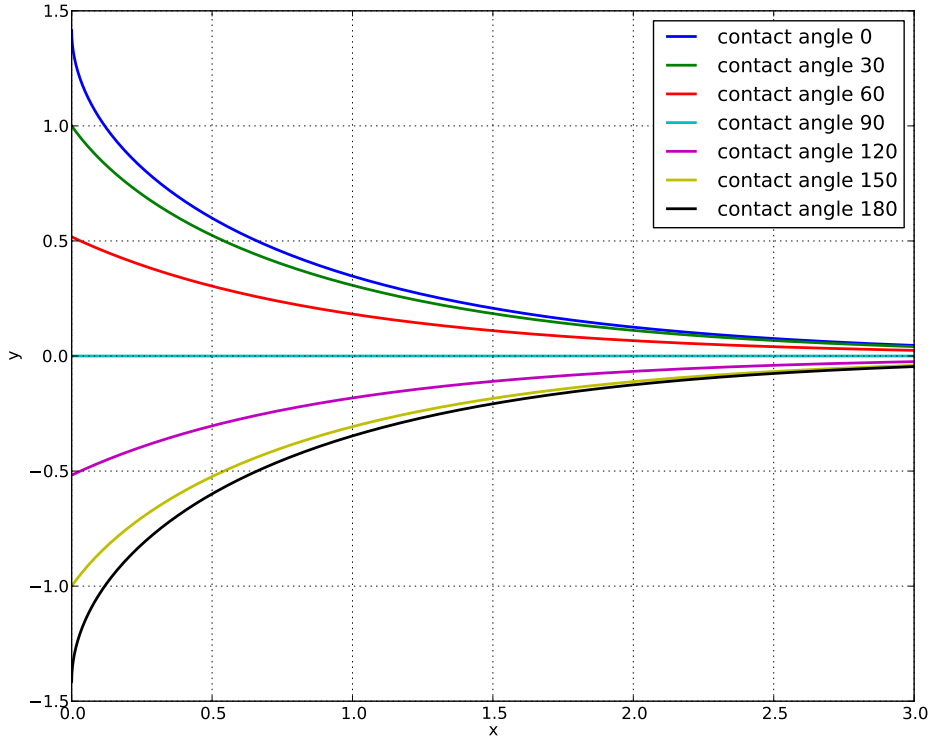


Figure 4.3: Flat-wall meniscus for all contact angles in steps of 10 degrees. Close to the wall, the menisci start with the contact angle; far from the wall, they approach the x-axis exponentially [56].

To extend the solution to inclined walls, the method exploits the translational invariance and simple trigonometric functions (Fig. 4.5). If one tilts the wall until the correct contact angle is reached, then no correction is needed. If one tilts the wall further, correction is needed in the opposite direction. The correction length d in Fig. 4.4 is perpendicular to the fluid surface. It is a function of the difference between the meniscus heights from Eq. 4.1 for the tilt angle of the wall, β, and the desired contact angle α:

$$d = l\left(2\sin\frac{\pi - 2\alpha}{4} - 2\sin\frac{\pi - 2\beta}{4}\right). \tag{4.2}$$

To compensate for the tilt the length c_0 is:

$$c_0 = \frac{d}{\sin\beta}.$$

The correction vector $\mathbf{c}$ that moves the contact line has length c_0, is parallel to the collision surface, and lies in the plane of the collision surface normal and

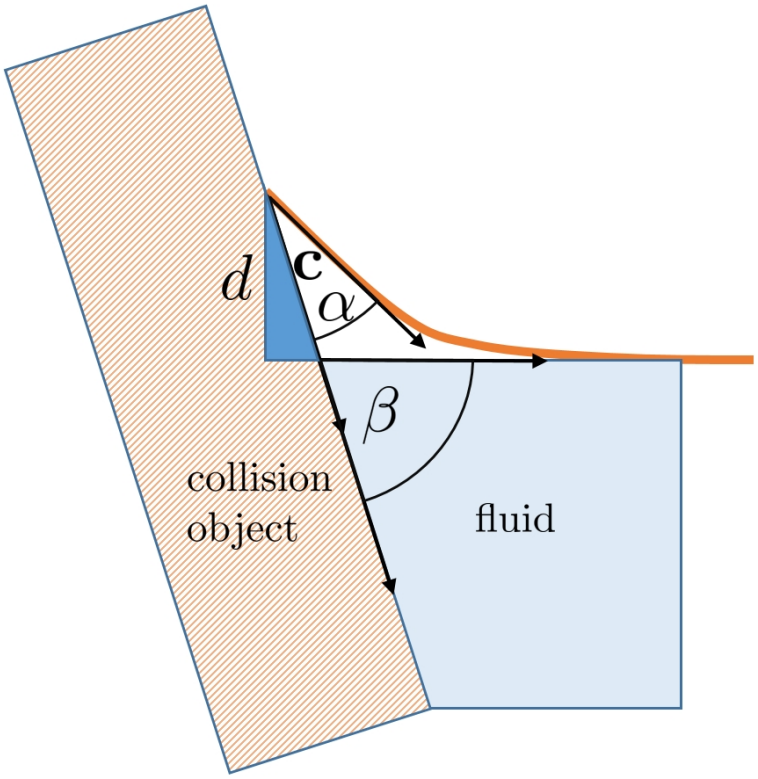


Figure 4.4: Contact angle α, tilt angle of wall β, and the correction length d.

the fluid normal. To calculate the direction of the correction vector $\mathbf{c}$ the two normal vectors and their scalar product can be used to obtain:

$$\mathbf{c} = c_0 \frac{\mathbf{n}_{\text{fluid}} - \mathbf{n}_{\text{collision}}(\mathbf{n}_{\text{fluid}} \cdot \mathbf{n}_{\text{collision}})}{||\mathbf{n}_{\text{fluid}} - \mathbf{n}_{\text{collision}}(\mathbf{n}_{\text{fluid}} \cdot \mathbf{n}_{\text{collision}})||} .$$

Once the correction vector is calculated with the correct length from the analytical solution, the ray intersection function that is used for direct raytracing of the fluid surface can be modified.

The presented method works for any kind of implicit description of fluid surfaces with distance fields. The direct and efficient support for grid-less, particle-based simulation methods is its main strength. For such applications, a modified version of the Zhu-Bridson [112] surfacing method is used that creates a distance field by averaging the positions of the nearby particles taking the kernel weight into account, as can be seen in Fig. 4.6. The radius of each particle is also averaged using the kernel weight. The result is a distance field for the fluid surface. Based on the distance to the collision object the average radius $\bar{r}$ can be offset with the meniscus correction length d solution from Eq. 4.2.

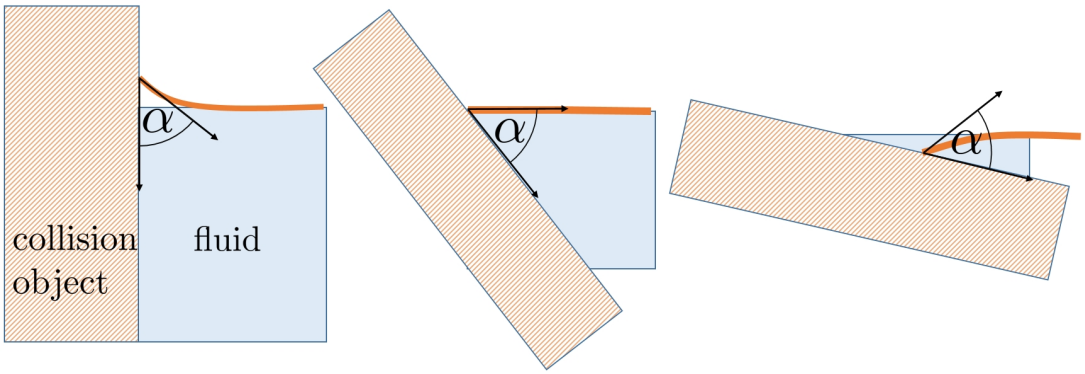


Figure 4.5: Meniscus at different wall inclinations.

In this way, the same curve can be obtained in the case of a horizontal surface at a vertical wall, i.e., a fully correct solution. In other scenarios, the method creates a plausible solution because the correct collision angle is guaranteed at the collision object, and the rest of the interface behaves smoothly before it finally reaches the original position of the interface from the fluid simulation. The additive nature of the resulting distance fields automatically smoothens out problems of self-intersection that mesh-based approaches would have, when two solids are close to each other.

4.5 MENISCI IN THE SPH SETTING

The method is independent of the simulation method as long as the results of the simulation yield a fluid representation based on distance fields. A prominent example of a particle-based fluid simulation is SPH as introduced in Chapter 2. This section describes how the generic meniscus-generating rendering approach can be used in the context of SPH simulation. The method assumes that the collision object is represented by its boundary surface in the form of a triangle mesh. Here, methods can be used that create distance fields from triangle meshes by computing the signed distance transform (e.g., Bærentzen and Aanæs [13]). By setting up distance fields for the collision surfaces, there are now two distance fields that can be used to create the final fluid surface.

There is only a minimal preparation step before rendering. In this step, acceleration structures for the particles are built. The particles are stored into a bounding volume hierarchy (BVH) and collision objects are transferred into distance fields.

All work takes place in the ray intersection step while rendering. The intersection is found by raymarching. As a first step of the ray intersection, the

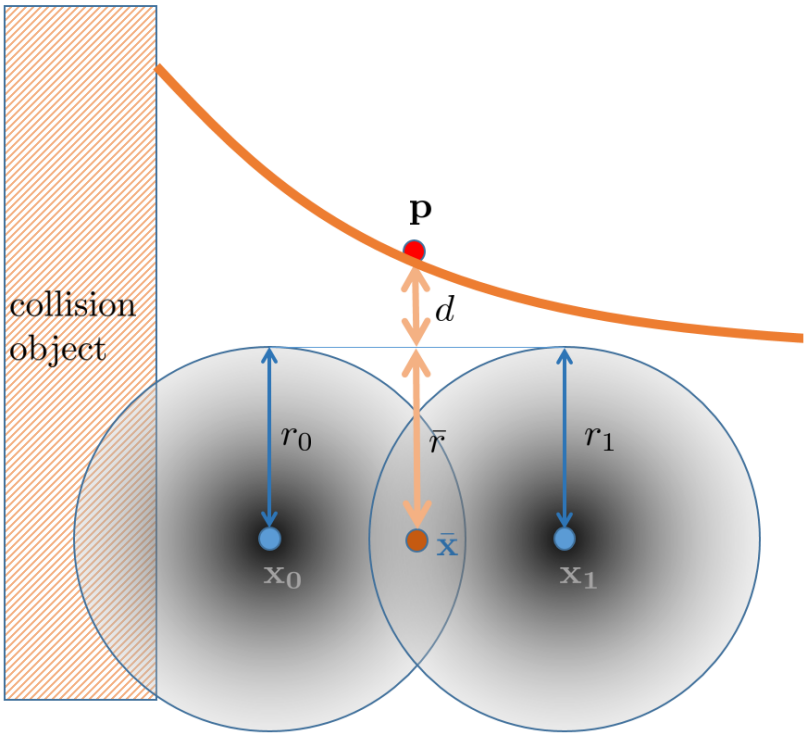


Figure 4.6: The analytic meniscus is added to the average radius $\bar{r}$ from the Zhu-Bridson algorithm.

algorithm finds the intersected leaf nodes of the BVH. It then sorts those nodes by their intersection point along the ray. The routine now marches along the ray from the first intersection point into bounding boxes and evaluates a distance function $\phi(\mathbf{x})$ for the particles in the leaf node.

As can be seen in Fig. 4.6, the base distance field for a fluid surface can be created with the Zhu-Bridson algorithm that leads to a signed distance field around the particles by using a weighted average of the nearby particle positions $\mathbf{x}_i$ and the weighted average of their radii r_i. The SPH kernel is denoted W_i, h is the smoothing length, and k is a suitable kernel function that smoothly drops to zero. The corresponding equations read:

$$\bar{\mathbf{x}} = \sum_i W_i \mathbf{x}_i \,,$$

$$\bar{r} = \sum_i W_i r_i \,,$$

$$W_i = \frac{k(||\mathbf{x} - \mathbf{x}_i||/h)}{\sum_j k(||\mathbf{x} - \mathbf{x}_j||/h)} \,.$$

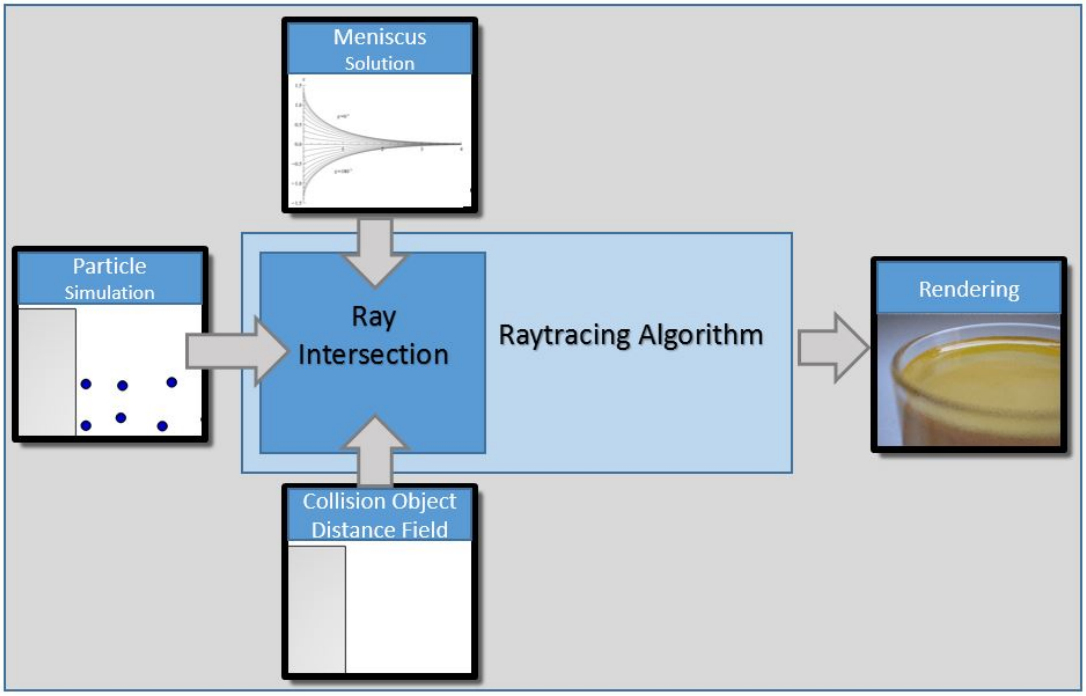


Figure 4.7: Ingredients needed at render time in the ray intersection step.

The meniscus correction length d from Eq. 4.2 is added to the weighted average of their radii r_i to obtain the final distance function $\phi(\mathbf{x})$:

$$\phi(\mathbf{x}) = ||\mathbf{x} - \bar{\mathbf{x}}|| - (\bar{r} + d) \, .$$

Given the signed distance fields for the fluid surface and the collision surface, there is enough information to build the surface at render time, as shown in Fig. 4.7. Additionally, the fluid's distance field can be restricted to the outer side of the distance field of the collision objects. In most cases, the fluid simulation already took care of this through its collision detection. However, the resulting surface can be enhanced with the help of the distance fields. The two fields can be combined with a minimum function.

This distance function is used by the ray marching step to find the intersection. The same procedure is used for primary rays and secondary rays (shadow rays, global illumination rays, reflection rays, etc.) The geometric normal vector of the fluid surface is derived from the distance field. An alternative method implements normal mapping for the case of the meniscus modification: it keeps the original Zhu-Bridson surface and only perturbs the surface normals to fit the meniscus normals from the analytic solution. For comparison, this was implemented as shown in Fig. 4.9. It can be assumed that in most use cases a meniscus will hardly change the silhouette of the fluid but the highly varying normal of the reflective fluid surface causes a big visual difference. This

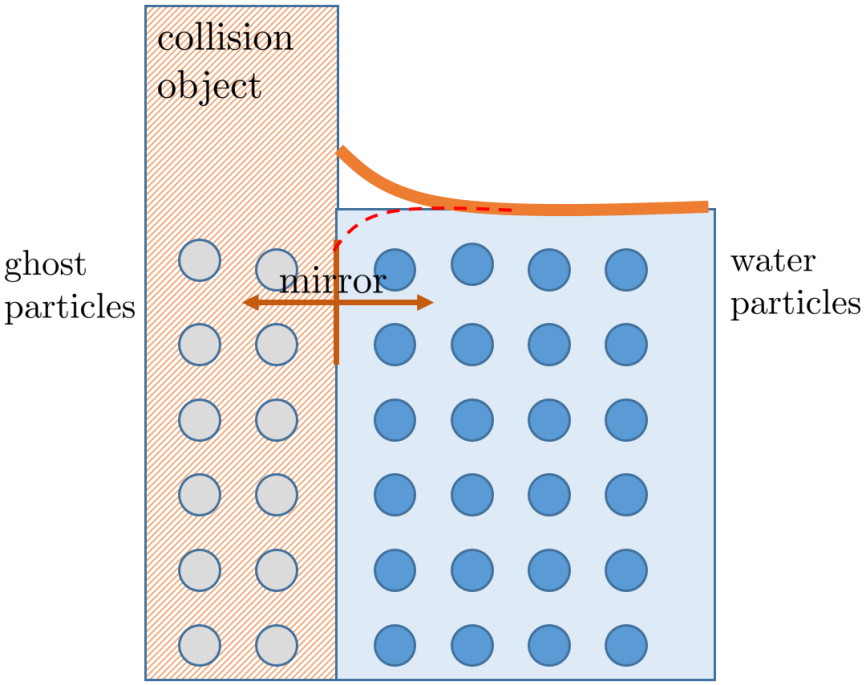


Figure 4.8: Ghost particles are created to ensure a continuous surface near the collision object. The dashed line shows the surface without ghost particles.

approach has its limit at grazing angles. Here, the normal mapping approach cannot sample the highlights sufficiently where a real displaced meniscus covers more pixels and therefore has enough ray samples for the highlights.

There are other surfacing methods for fluids, including [4], [112], [86], [106]. Research showed that all surfacing methods have to deal with mass conservation at the border (for rendering purposes). Most surfacing methods use a filter kernel that will have a radius that is 3–4 times the particle radius. This leads to a smoothing of the sharp edge at the fluid–solid interface. Although no meniscus correction is applied, this creates a negative meniscus that would be the result of a contact angle greater than 90 degrees, which is noticeable in close-up renderings. For an exact meniscus correction, the presented method needs a clean 90-degree contact angle in the case of a vertical wall. To ensure a straight angle the method creates a thin layer of ghost particles inside the collision object that will extrapolate the surface far enough, as can be seen in Fig. 4.8. Even if no meniscus correction is applied, the use of ghost particles can at least ensure a neutral contact angle of 90 degrees.

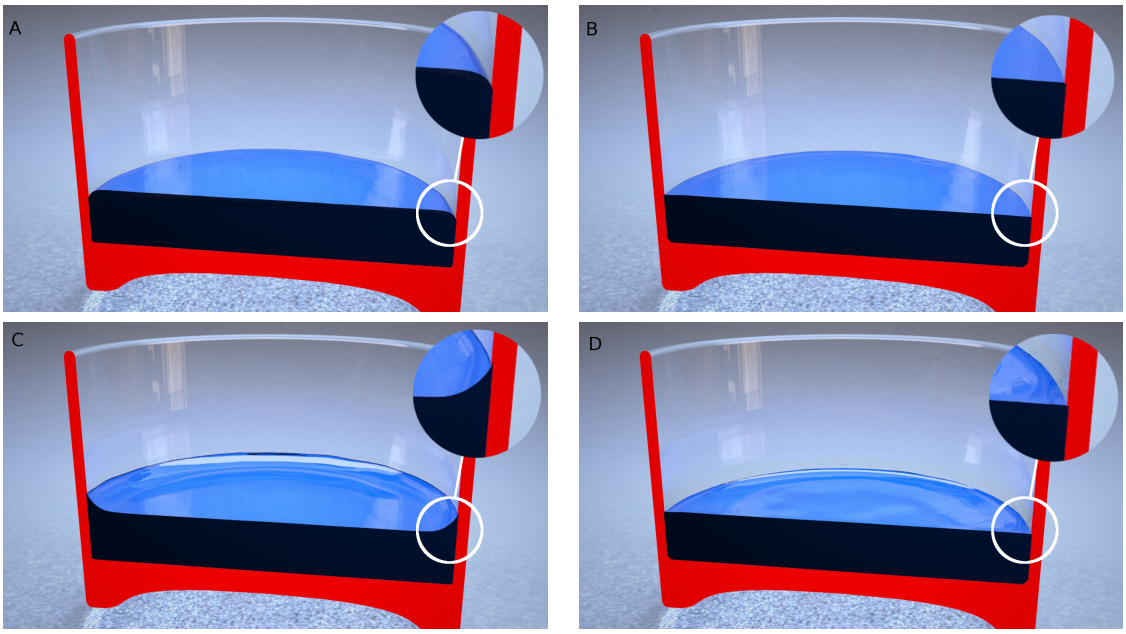


Figure 4.9: Top to bottom: (A) Surface with no ghost particles, (B) surface without meniscus correction, (C) geometric meniscus correction, (D) normal mapping with modified surface normals alone. Please note that on the image (D) you obtain the highlight at the border although there is no geometric difference to the surface without meniscus. A clipping plane is used in all images; the cutout surface is marked red.

4.6 IMPLEMENTATION AND RESULTS

This section reports on the computational performance of the presented technique and the quality of the resulting fluid interface.

The computation time for 3 different scenes was measured. The algorithm was implemented into the commercial 3D package 3ds Max 2016 with the 3ds Max SDK using C++ and Open Multi-Processing (OpenMP) for multi-threading. Our measurements were performed on a laptop computer with an Intel i7-2670QM CPU at 2.20 GHz with 4 cores (8 logical processors). The scenes were set up with the GPU-based SPH implementation from Chapter 3. The implementation of the method allows for fast adjustments (and artistic variations) since adding the meniscus is only a post-processing step to a cached fluid simulation.

The implementation uses the OpenVDB framework [68] for the distance field generation. The raytracing method builds upon the same method that was used in Chapter 3 for direct raytracing of isosurfaces and was integrated into

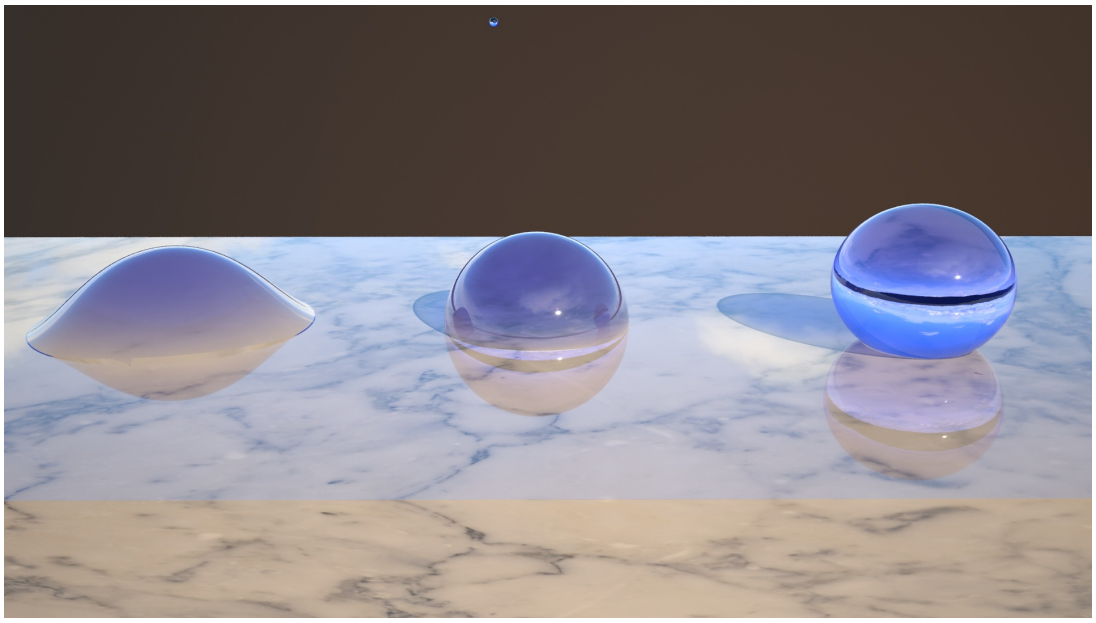


Figure 4.10: Deformation of spherical drop based on our closed-form solution for different contact angles caused by varying material properties (left to right: high wettability to low wettability). Note in particular that a single SPH particle is used at the same height in each of the three cases.

the commercial raytracing engine V-Ray version 3.3, which shows that the method lends itself to simple integration in existing rendering systems.

In Table 4.1, render times between the described method and a mesh-based approach (using Marching Cubes) for rendering are compared. The tested scenes can be seen in Fig. 4.15. The "cup" and "boat" scenes were rendered with 1280×1024 pixels. Please note that the corridor scene with the highest complexity was rendered with a lower resolution of 640×480 pixels. All images were rendered using fixed-rate antialiasing of 1 sample per pixel, simple lighting, and gray materials, as only the timings of the ray intersection algorithm are relevant. Sophisticated antialiasing schemes create an unpredictable number of samples and therefore falsify the timings. The time added by the meniscus correction was 50% for the simple scene, 9% for the medium scene, and negligible for the complex scene. The main reason for the extra time is the additional preparation step to create the distance field with OpenVDB. The ratio of time needed for building the collision distance field, to the actual rendering, is higher for scenes with lower particle counts. It can be seen that direct raytracing is faster in situations with high particle and polygon counts. In these cases, the Marching Cubes step of mesh-based approaches adds significant preparation time. The relatively small meniscus puts high requirements on the needed mesh resolution. This leads to even longer meshing times.

Table 4.1: Different computing times for the performance test scenes. Columns show timings in minutes:seconds except for the columns particle count and vertex count.

Scene	**Particle count**	**Direct raytracing**	**Geometric meniscus correction**	**Marching Cubes**	**Vertex count**
Simple scene (fluid in cup)	20 367	0:36	1:13	0:26	36 624
Medium complexity (boat in water)	97 590	3:06	3:06	0:41	4 700 000
High complexity (corridor)	179 000	1:17	1:25	3:05	1 300 000

Direct raytracing is especially helpful for small render tests without antialiasing and for complex scenes. Here, it outperforms methods with meshing. This allows fast iterations in the lighting phase of a computer graphics production. The method scales better with increasing complexity of the scene than increasing the number of ray intersections for more pixels or more secondary rays. Depending on how many ray intersections per pixel have to be made, the preparation step of a mesh-based rendering pays off.

The effect of the contact angle for droplets on various flat surfaces having different properties was rendered with the presented method in Fig. 4.10. Fig. 4.10 already shows the power of our approach: the underlying simulation works on a very coarse level—it models a drop by a single particle from SPH [62]. The visible differences between the three drops in Fig. 4.10 are exclusively modeled and computed by our technique at render time. Fig. 4.9 demonstrates the difference between a surface without ghost particles (A), a surface with ghost particles but without meniscus correction (B), and a surface with our meniscus generated at render time (C). Please note the insets that magnify the shape of the boundary for each variant. Fig. 4.11 compares the meniscus at the sidewall of a glass with a real photography. The adaptive nature of the direct raytracing ensures a smooth reflection. The rendering also shows that our method supports complex shader effects like subsurface scattering. Fig. 4.12 shows the geometric nature of the surface and how the meniscus reacts to differently inclined walls. Fig. 4.13 further illustrates the quality of our approach. As in Bourque et al. [18], the method can simulate the correct caustics like the "shadow sausage effect" but also capture the highlight at the straw. Again, our approach is compared to the real photograph of the setting in Lock et al. [59]. Fig. 4.14 is a frame from an animation. The method has good temporal coherence, which leads to a smooth, flicker-free movement of the surface. This can also be seen in the video in the video appendix [63]. The comfort of not having to create and store geometry caches before rendering animations is a big advantage of direct raytracing methods.

The method with a geometric meniscus is also compared to a version where

only the surface normals are changed. Images produced by the two versions are shown at the bottom of Fig. 4.9. Experience showed that the computational costs are similar for both variants; for example, the boat scene took 3 min 26 s for normal mapping as opposed to 3 min 6 s for the geometric correction (see Table 4.1). Surprisingly, the normal map approach is not faster. The explanation is that the overhead in the ray intersections part is computationally not more expensive than the other operations on calculating the normals.

This chapter showed that by analyzing the meniscus effect in detail and deriving a closed-form solution, a method can be found to combine an analytic solution with data from a fluid simulation. Without storing highly tesselated meshes, fine detail to coarse simulations can be added at render time.

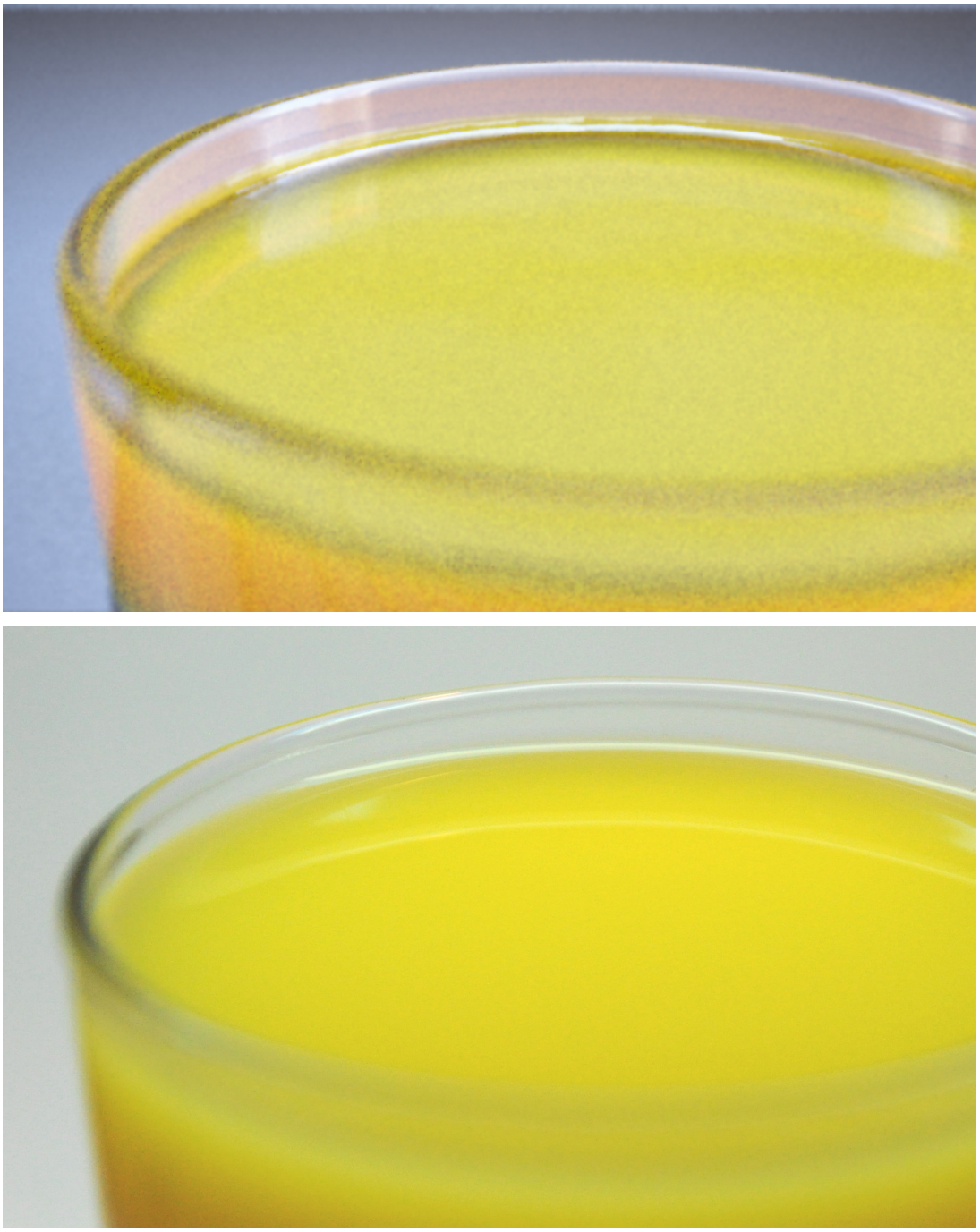


Figure 4.11: The meniscus at a sidewall, computed with our technique (top) and, for comparison, in a real picture (bottom).

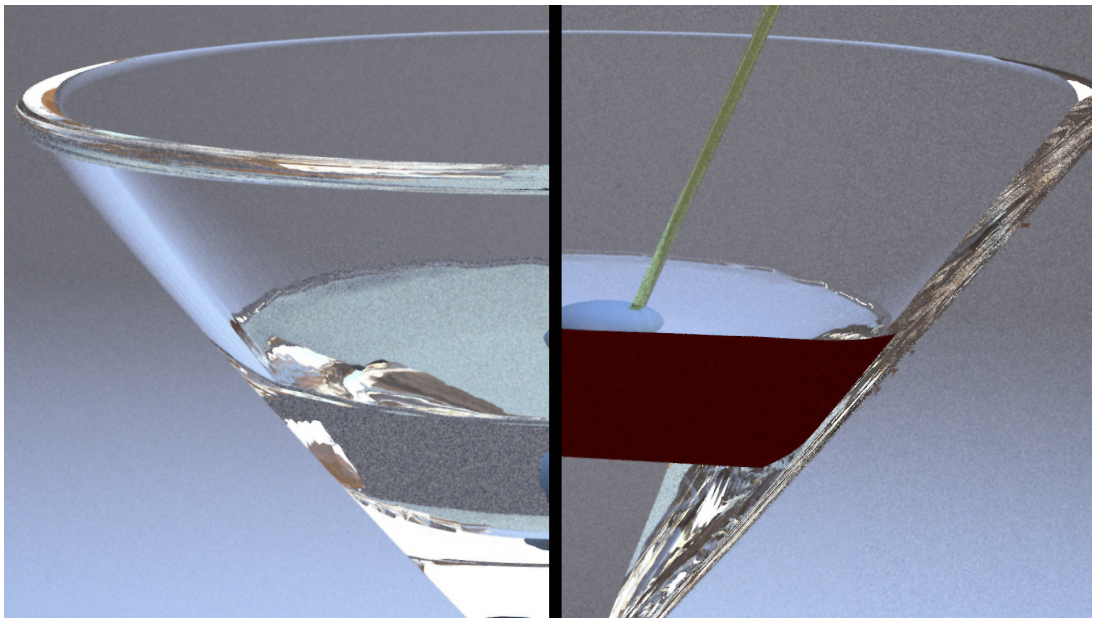


Figure 4.12: "Just a drink, a Martini, shaken not stirred": the meniscus at a slanted border. It is less visible since the contact angle is almost reached with the wall inclination. The right side is rendered with a clipping plane to show the shape of the meniscus.

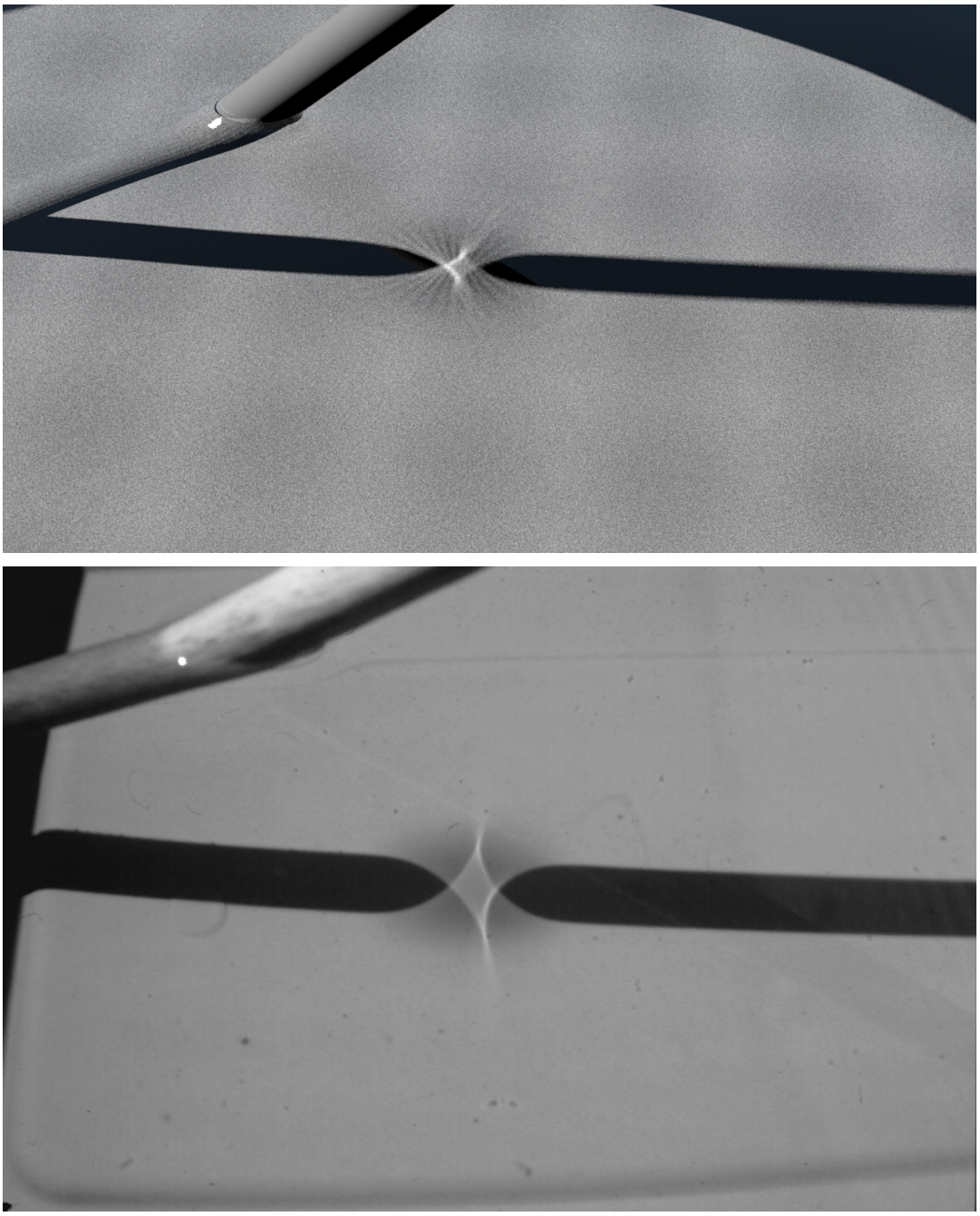


Figure 4.13: Our treatment of the meniscus forming (top) creates correct caustics such as the "shadow sausage effect". Here, a straw is dipped into water. For comparison, the bottom image shows a real photograph reprinted from Lock et al. [59].

Figure 4.14: Frame from an animation. The meniscus is consistent throughout the animation and works well even in cases where our closed-form solution for a vertical wall is not completely adequate.

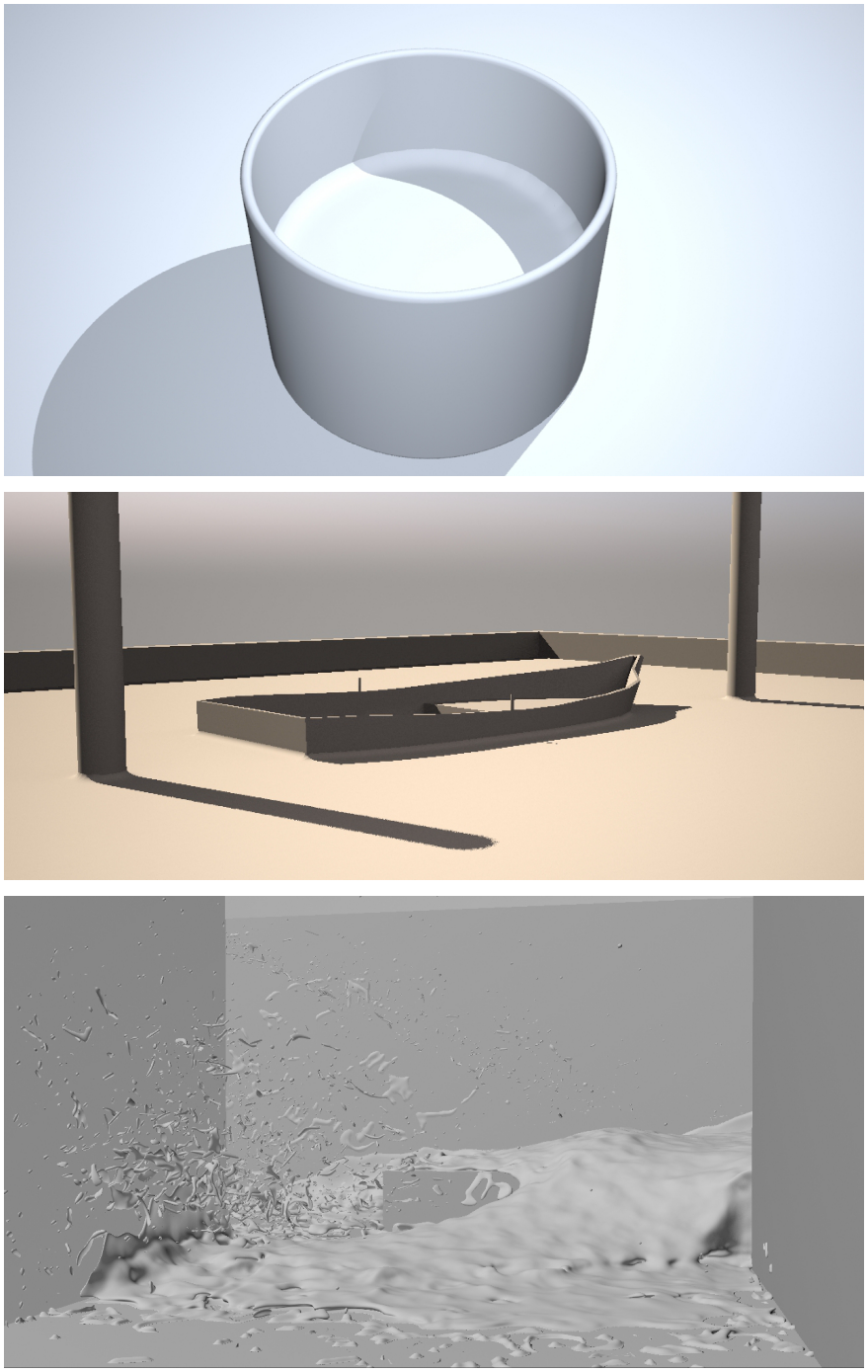


Figure 4.15: Test scenes for performance measurements. Top to bottom: simple complexity scene of fluid in a cup, medium complexity of a boat in water, high complexity scene of fluid in a corridor.

CHAPTER 5

Efficient 2D Simulation on Evolving 3D Surfaces

The previous chapter addresses a specific small scale effect with an analytic approach that is only applicable for this exact effect. The presented method is not applicable to other kinds of small scale effects. This chapter establishes a more generic method to achieve secondary effects on moving surfaces. It can be used to simulate fluid flow on evolving surfaces, e.g., an oil film on a water surface, but also for other kinds of effects that can be described with a PDE. Given an animated surface (e.g., extracted from a particle-based fluid simulation) in 3D space, a second simulation on the input surface is added. In general,

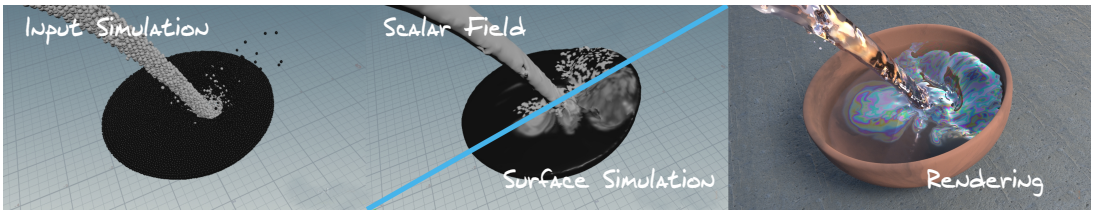


Figure 5.1: Water polluted with oil is poured into a cup. The oil film is simulated on top of the existing fluid simulation. The velocity and mass are taken from the existing animation. On the left is the coarse input simulation. In the upper half of the middle image is the resulting scalar field. The lower half shows the result of the method. The high-resolution 2D simulation adds convincing visual details to the coarse input simulation. The right image displays the final rendering that needs the fine details from the surface simulation to generate the high-resolution thin-film interference effects.

the proposed method solves a PDE on a level set surface obtained from the animated input surface. The properties of the input surface are transferred into a sparse volume data structure that is then used for the simulation. Coupling strategies between input properties and the simulation are introduced, and conservation of mass and momentum is added to existing methods that solve a PDE in a narrow-band using the closest point method (CPM) [77]. In this way, the method efficiently computes high-resolution 2D simulations on coarse input surfaces. This approach helps visual effects creators easily integrate a workflow to simulate material flow on moving surfaces into their existing production pipeline.

In typical workflows for generating digital visual effects, a team of VFX artists iteratively refines a given sequence until they achieve good quality. Going from rough storyboards over blocked animation to the final shot with many layers of physical simulations, each shot passes through the VFX pipeline, where domain specialists add new effects and details. With physical simulations, often an effect is finished and approved before secondary effects are layered on top of it. For example, after simulating the fluid flow of a water surface, secondary effects like splashes and foam [93] [43] are added on top of this "basic" water simulation, which will be referred to as "base simulation" in this work. In this context, a method is proposed to add secondary effects on top of the base simulation by solving a PDE on the surface. As an example, a thin-film 2D fluid simulation is simulated on top of a possibly precomputed, bulk fluid simulation as shown in Fig. 5.1. The method employs a one-way coupling to transfer momentum and mass from the 3D fluid simulation to the surface simulation. The coupling is based on physical derivations and plausible parameters are provided for it to control its effects. This coupling allows iterating on the secondary effects with a consistent high-resolution 2D simulation on top of the unchanged coarse 3D input simulation.

To provide a generic tool for different kinds of small scale effects, the goal is to solve a PDE on a moving 2-manifold. The approach can then be used to address different types of problems that require solving PDEs like fluid flow, reaction-diffusion texture synthesis, or 2D wave equations. Fig. 5.2 depicts the steps of the approach. It starts with a moving surface as input geometry. After converting the input data into a distance field and transferring values into a narrow-band grid around the surface, the method introduces quantities from the input 3D simulation into the surface domain in a coupling step where the strength of the coupling can be driven by parameters. Then, CPM is used to embed the 2-manifold in the 3D space of the moving surface and solve 2D PDEs in a 3D narrow-band.

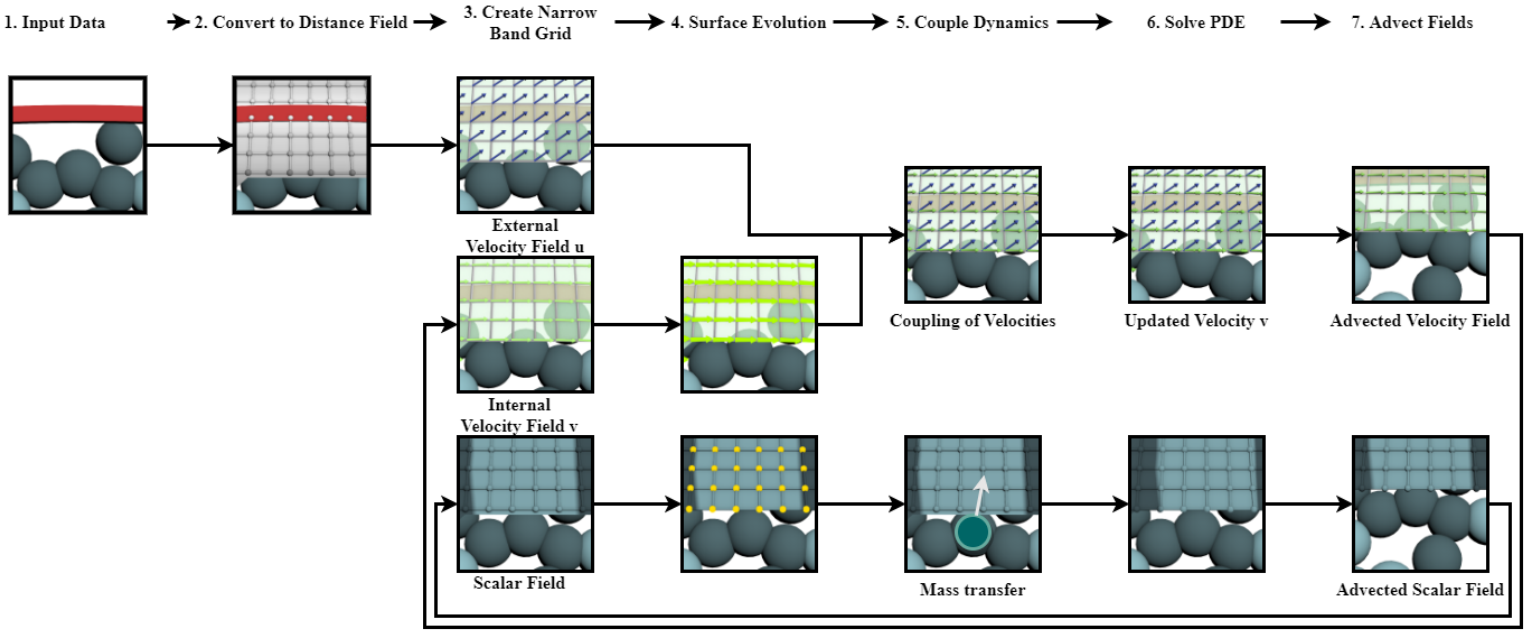


Figure 5.2: The seven steps of the presented method as a flow chart. The steps are placed in order of execution from left to right. After the input of the external data (Step 1), the data is transferred into the narrow-band (Step 2 and 3). Next, the surface is evolved (Step 4) and the outer process is coupled with the inner dynamics (Step 5), before solving the set of PDEs on the surface (Step 6). Finally, the fields are advected (Step 7).

With the proposed method, secondary effects can be modeled very efficiently. The approach is evaluated by simulating a variety of effects such as pouring an oil film into water, simulating reaction-diffusion on a water surface, or the surfactants in a heated soap bubble.

The code is available as an open source project on GitHub [65] to simplify the integration into existing production pipelines.

5.1 RELATED WORK

Secondary effects can be simulated on top of the base simulation in two different ways. The first way is to directly integrate them into the simulation, for example, bubbles in foam [96], [24], [45]. Here, the secondary effects are two-way coupled with the main fluid. Modeling the secondary effects in such a way does not allow for post-processing of existing simulations and animations without the need to re-simulate. To address this problem, the surface simulation is decoupled from the input simulation. Furthermore, this decoupling allows one to use a higher resolution on the surface simulation and adds fine details on top of coarse inputs.

The second way to add additional effects like splashes, bubbles, and foam is to simulate them with the main fluid but not to exert forces back onto the main simulation [93], [43], [50]. Splashes and bubbles need true 3D solvers as they extend into, or out of, the surface. Foam only exists on the surface and can be rendered by tagging surface particles of the original simulation [8] or by moving texture patches with the surface [31]. For this work, effects that are limited to the 2D surface are of interest, but the goal is to simulate a finer resolution on the surface than the original 3D fluid simulation offers.

Solving PDEs on 2-manifolds has been addressed earlier, e.g., Stam [89] simulated fluid flows on static Catmull-Clark surfaces and Shi et al. [82] on static polygon meshes. Using the CPM, this can also be computed in realtime [10]. The theory for solving PDEs on evolving surfaces is explained by Xu and Zhao [103]. One example is the simulation of soap film on bubbles [46] where the PDE is solved on a polygon mesh or solving the material flow within a soap bubble on a staggered spherical grid [40]. On evolving surfaces, Mercier et al. [61] solve a PDE in a 3D narrow-band grid using the CPM to enhance the original simulation with additional turbulence. Closest to the presented approach is the Semi-Lagrangian Closest Point Method [11]. In comparison to existing CPM-based methods, equations for conservation of mass are added. This allows one to correctly adapt scalar values when the overall surface area changes. The inner velocity is also adjusted based on the surface movement. Furthermore, a one-way coupling is modeled using mass and momentum trans-

fer to create a plausible linkage between the evolution of the input surface and the behavior of the resulting 2D simulation.

5.2 MODEL

This section describes how the physics on the surface is modeled. Furthermore, some example use-cases are presented with different underlying physical phenomena.

5.2.1 Overview

The starting point is an evolving surface M. This surface typically results from a simulation, a keyframe animation, or any other kind of time-dependent process. This process that evolves the surface will be called the outer process. Next, another process is defined on M (e.g., the simulation of a substance) by a set of PDEs and is coupled to the outer process that is evolving M. An example of such a process could be a thin sheet of oil floating on an evolving water surface as shown in Fig. 5.1. For a better understanding of the approach, the example of a thin oil sheet on water will guide us through the overall system in the next subsections. The process can be split into three aspects.

The first aspect is the **evolution of the surface** M due to the velocity field $\mathbf{u}$. In the following, the surface evolution characterized by $\mathbf{u}$ will be referred to as the outer process and the process that happens one on the surface will be called inner dynamics. The surface space on which the dynamic process is modeled changes with the surface evolution. To account for this change, the method must define how quantities evolve on the surface. In our example with the oil, it describes how the oil is transported in space when the water is moving in normal direction. In Subsection 5.2.2, an operator O is defined that will describe how the needed scalar and vector quantities that define the oil can be transported with the surface.

The second aspect is the **coupling** that determines how the outer process affects the one on the surface. In our example, it accounts for the movement of the water in tangential direction, i.e., the acceleration of the oil due to friction forces. In Subsection 5.2.3, equations that describe the coupling for both scalar and vector quantities are established.

And finally, the third aspect is the **modeling of the dynamics on the surface**. This part defines how the secondary dynamic process is modeled on the surface, i.e., how a set of PDEs is solved on the surface, in our example, the dynamics of the oil itself (by taking the evolution of the water into account). The approach and examples of different dynamic processes are presented in Subsection5.2.4.

5.2.2 Evolution of the Surface

Since the surface $M \subset \mathbb{R}^3$ evolves by a velocity field $\mathbf{u} := \mathbf{u}(\mathbf{x}, t)$, the surface is a function of time and can be written as $M(t)$, where t denotes a certain point in time and Δt a small time increment. Therefore, $M(t_0 + \Delta t)$ results from $M(t_0)$. For convenience, M will be written instead of $M(t_0)$ and M' for $M(t_0 + \Delta t)$. The space $\mathcal{M}$ is used to describe the set of points on the surface together with attached scalar and vector quantities in the tangential space and is defined by:

$$\mathcal{M} = \bigcup_{\mathbf{p} \in M} \{\mathbf{p}\} \times \mathbb{R} \times T_{\mathbf{p}} M, \tag{5.1}$$

where $T_{\mathbf{p}} M$ denotes the tangent space specified by the surface normal $\mathbf{n}(\mathbf{p}, t)$ at point $\mathbf{p}$. To put this in the context of our oil example, $\mathcal{M}$ is the set of information that describes oil on water.

A basic requirement for modeling a dynamic process on M is an unambiguous mapping of every surface point $\mathbf{p} \in M$ between time steps. A map O is constructed that describes the evolution of points and their attached quantities due to the outer process. The map O relates points between M and M' and the attached scalar quantities, $a(\mathbf{p}, t) \in \mathbb{R}$, as well as vector-valued quantities, $\mathbf{v} := \mathbf{v}(\mathbf{p}, t) \in T_{\mathbf{p}} M$. Typical quantities would be, e.g., mass density and velocity when modeling fluid flow on the surface. In the following, only one scalar and one vectorial quantity are exemplarily noted in the equations, but it is possible to connect an arbitrary number of quantities to $\mathbf{p}$ in the same way.

The quantities needed for the inner dynamics can be described as elements of $\mathcal{M}$. The map O relates elements from $\mathcal{M}$ to elements from space $\mathcal{M}' = \bigcup_{\mathbf{p}' \in M'} \{\mathbf{p}'\} \times \mathbb{R} \times T_{\mathbf{p}'} M'$:

$$O : \mathcal{M} \to \mathcal{M}', \begin{pmatrix} \mathbf{p} \\ a \\ \mathbf{v} \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{p}' \\ a' \\ \mathbf{v}' \end{pmatrix}. \tag{5.2}$$

An illustration of the map O is given in Fig. 5.3.

The influence of the outer process characterized by $\mathbf{u}$ is separated into normal $\mathbf{u}_n$ and tangential $\mathbf{u}_t$ components. The adapted velocity $\tilde{\mathbf{u}}_{\mathbf{n}}$ is defined as $k\,\mathbf{u}_{\mathbf{n}}$, where $k \in \mathbb{R}$ has to be chosen in such a way that $\mathbf{p}' \in M'$ holds. In other words, the model ensures that when a point $\mathbf{p}$ is moved to its new position $\mathbf{p}'$, it always sticks to the surface. The tangential component is used to model friction between the outer and inner dynamics and is discussed in Subsection 5.2.3. For the map O, it is assumed that the dynamics are coupled without friction and, therefore, $\mathbf{u}_t = 0$ and the outer process does not affect the inner dynamic,

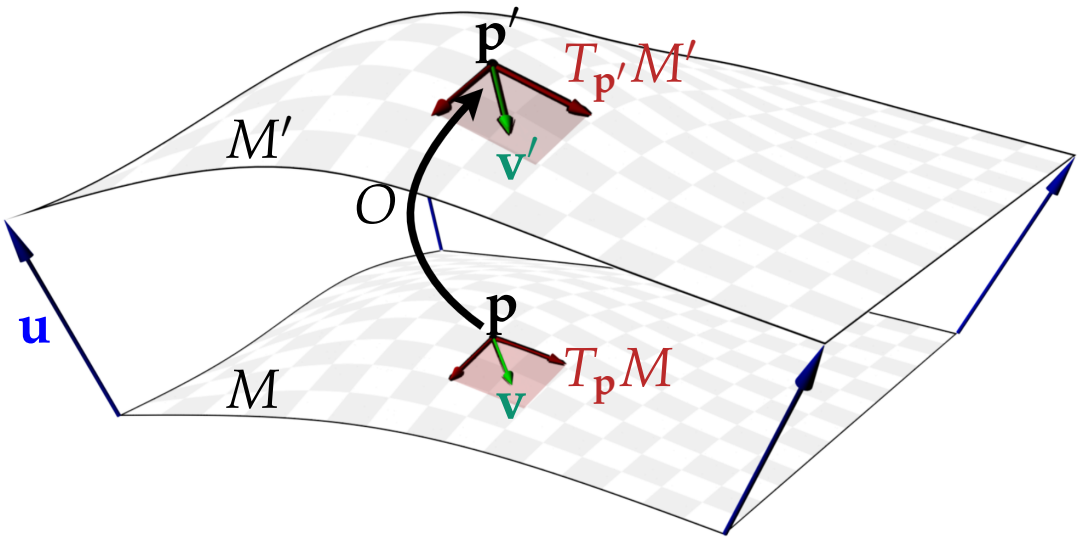


Figure 5.3: Illustration of the map O that relates $\mathbf{p} \in M$ and $\mathbf{p}' \in M'$ and also maps tangent space $T_{\mathbf{p}}M$ to $T_{\mathbf{p}'}M'$.

i.e., $m \in \mathcal{M}$ is only altered by $\mathbf{u}_\mathbf{n}$. Consequently, points are transported with the altered normal part of the outer velocity $\tilde{\mathbf{u}}_\mathbf{n}$. This already forms the first part of our map O that defines the relation of points on the surface between time steps.

Now, that the relation of surface points between time steps is defined, scalar quantities can be attached to the points. However, some quantities need special attention. Scalar quantities can be distinguished into two types: intensive and extensive ones [75]. An extensive property is a global property (e.g., mass or volume). Such a property is only advected alongside $\mathbf{p}$ and not changed, i.e., $\frac{D_O a}{Dt} = 0$. In contrast, if a describes an intensive physical property, such as density, if the surface diverges, the concentration of a decreases and if the surface converges, the concentration increases.

An example of the change of an intensive scalar property is a growing and shrinking sphere. If a quantity like density is attached to the sphere's surface, the density is expected to decrease as the sphere grows and increase when the sphere shrinks. This can be determined by looking at the divergence of the velocity field. The divergence is positive when space is growing and negative when it is shrinking. Consequently, the divergence of the velocity field in surface space $\nabla \cdot (\tilde{\mathbf{u}}_\mathbf{n})_{T_\mathbf{p}M})$ can be used to describe the second part of the map O:

$$\frac{D_O a}{Dt} = -a(\nabla \cdot (\tilde{\mathbf{u}}_\mathbf{n})_{T_\mathbf{p}M}), \tag{5.3}$$

where $\mathbf{u}_{T_\mathbf{p}M} = (I - \mathbf{n}\mathbf{n}^T)\mathbf{u}$ is the velocity projected onto the tangential space

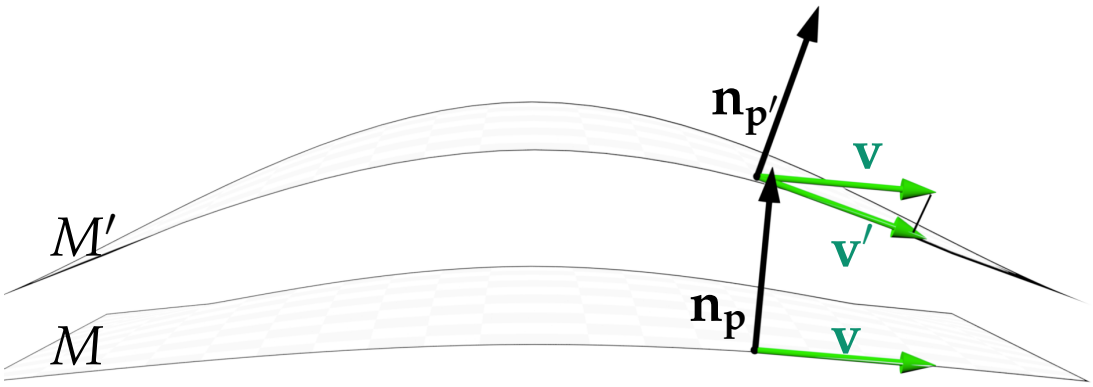


Figure 5.4: The map O transforms velocity vector $\mathbf{v}$ to vector $\mathbf{v}'$. First, $\mathbf{v}$ is advected to the point $\mathbf{p}'$. Next, it is projected back into $T_{\mathbf{p}'}M'$ and scaled to conserve momentum.

$T_{\mathbf{p}}M$ at point $\mathbf{p}$. A detailed derivation of the formula can be found in the appendix of Morgenroth et al. [1].

When advecting a vectorial quantity $\mathbf{v}$, as illustrated in Fig. 5.4, the method ensures that $\mathbf{v}' \in T_{\mathbf{p}'}M'$ holds after applying O, which means that the advected vector should be part of the tangential space for the next time step. This could be achieved by moving the vector along the point $\mathbf{p}$ and then projecting it onto the surface. However, this projection possibly changes the length of the vector. The presented model preserves the length of advected vector quantities.

Metaphorically, this means that $\mathbf{v}$ is advected and rotated back onto the surface. This intuitive image can be expressed as $\mathbf{v} \cdot \nabla \tilde{\mathbf{u}}_{\mathbf{n}} \mathbf{v} \frac{\mathbf{v}}{\|\mathbf{v}\|^2}$ as shown in a detailed derivation in [1].

If a is an intensive property, the combined material derivative $\frac{D}{Dt}O$ then reads as

$$\frac{D}{Dt}O = \begin{pmatrix} \tilde{\mathbf{u}}_{\mathbf{n}} \\ -a(\nabla \cdot (\tilde{\mathbf{u}}_{\mathbf{n}})_{T_{\mathbf{p}}M}) \\ \nabla \tilde{\mathbf{u}}_{\mathbf{n}} \mathbf{v} - \mathbf{v} \cdot \nabla \tilde{\mathbf{u}}_{\mathbf{n}} \mathbf{v} \frac{\mathbf{v}}{\|\mathbf{v}\|^2} \end{pmatrix}. \tag{5.4}$$

So far it was assumed that the outer process influences the inner dynamics only in normal direction. Next, it will be discussed how the tangential part $\mathbf{u}_t = \mathbf{u} - \mathbf{u}_n$ of the outer process will influence the inner dynamics.

5.2.3 Coupling

The combined material derivative for the map O ignores the tangential part $\mathbf{u}_t$ of $\mathbf{u}$. This is equivalent to a friction-less coupling, i.e., matter slides on

the surface and no adhesion is present. To mimic friction between the inner dynamics and the outer process, adhesive forces are modeled to reduce the relative velocities $\mathbf{v}^{\text{rel}} = \mathbf{v} - \mathbf{u}_t$ between the outer process and inner dynamics as

$$\frac{D_c\mathbf{v}}{Dt} = -s_1\mathbf{v}^{\text{rel}}, \tag{5.5}$$

where s_1 is a user-defined coefficient and $\frac{D_c}{Dt}$ denotes the rate of change induced by coupling effects. The vector $\mathbf{v}$ can be considered to be a small linear material line element exposed to the velocity $\mathbf{u}_t$. As derived in Morgenroth et al. [1], its rate of change would then read as $\left(\nabla_{T_\mathbf{p}M}\mathbf{u}_t\right)\mathbf{v}$. Both views are combined to model the total rate of change of $\mathbf{v}$ caused by coupling effects as

$$\frac{D_c\mathbf{v}}{Dt} = -s_1\mathbf{v}^{\text{rel}} + s_2\left(\nabla_{T_\mathbf{p}M}\mathbf{u}_t\right)\mathbf{v}. \tag{5.6}$$

The two coupling coefficients s_1 and s_2 define how strong the coupling is. The operator $\nabla_{T_\mathbf{p}M}$ is defined as $\nabla_{T_\mathbf{p}M} = (I - \mathbf{n}\mathbf{n}^T)\nabla$.

Coupling velocities between the outer process and the inner simulation is the most common case, but sometimes it would be desirable to transfer also scalar quantities. The transfer of intensive physical quantities (e.g., density) is modeled as sinks and sources. Inspired by Fick's laws of diffusion, the system as a whole will try to reach a state so that the concentration a^{out} from the outer process and the concentration a of the inner dynamics align. The speed of the reduction is determined by the factor s_3 and the formula for the change of concentration of the substance on the surface

$$\frac{D_c a}{Dt} = -s_3 a^{\text{rel}}, \tag{5.7}$$

to reduce the concentration difference $a^{\text{rel}} = a - a^{\text{out}}$. If only sources should be modeled, the concentration difference is restricted to negative values, i.e., $a^{\text{rel}} \leq 0$. Sinks can be modeled, when restricting $a^{\text{rel}} \geq 0$.

5.2.4 Modeling the Dynamics on the Surface

The last subsection discussed the surface evolution and coupling, whereas this subsection describes how dynamics on the surface can be modeled. Many physical phenomena on the surface can be modeled with a set of PDEs that operate on the scalar or vector quantities. A general description can be formulated as

$$\frac{D_s\mathbf{v}}{Dt} = F_1\left(t, \mathbf{v}, \partial, \ldots\right) \text{ and} \tag{5.8}$$

$$\frac{D_s a}{Dt} = F_2\left(t, a, \partial, \ldots\right), \tag{5.9}$$

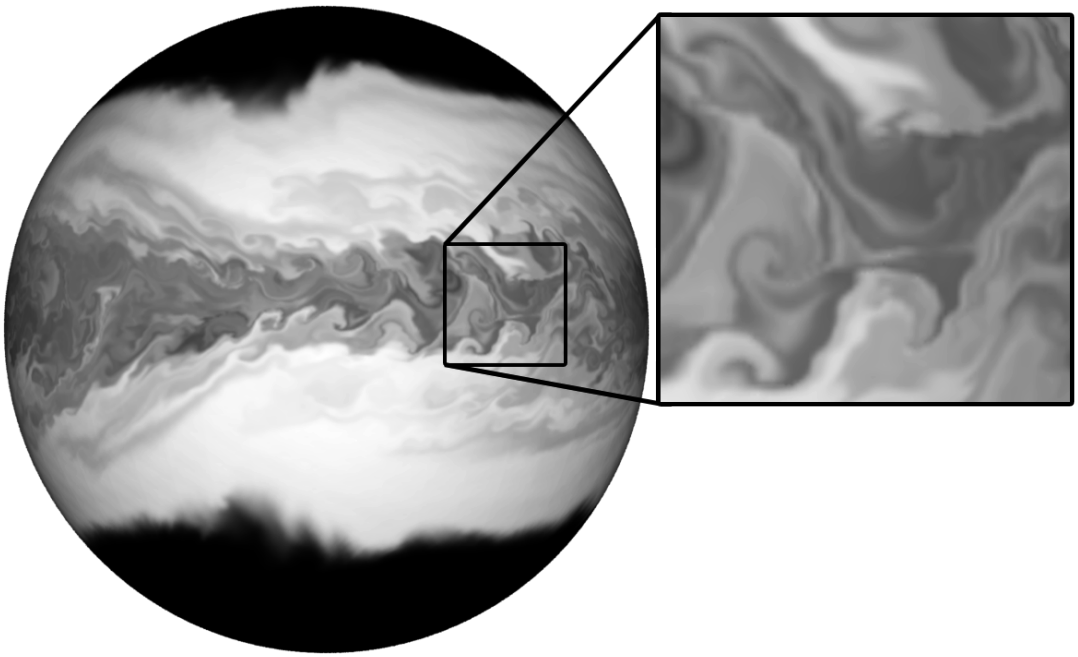


Figure 5.5: A rotating sphere with a fluid surface where the velocity of the internal simulation is coupled with the velocity of the sphere. Additional details are added with artificial vorticity and coupling noise.

where the functions F_1 and F_2 are placeholders for the set of PDEs for the respective phenomenon. For example, if fluid flow is modeled, F_1 could be the Navier-Stokes momentum equation and F_2 the continuity equation.

Eqs. 5.8 and 5.9 need special handling as they are not solved in 3D space but on the surface, i.e., this approach only allows for tangential deviations. In other words, the equations are solved locally in the corresponding tangent spaces $T_{\mathbf{p}}M$. To accomplish this, instead of using the ordinary spatial derivations, they are projected onto the surface, e.g., the operator ∇ is replaced by $\nabla_{T_{\mathbf{p}}M} = (I - \mathbf{n}\mathbf{n}^T)\nabla$.

Combining the surface evolution part from Section 5.3.4, the coupling part from Section 5.2.3, and the dynamics of this section, the complete governing equations for the presented method are then given by:

$$\frac{D\mathbf{v}}{Dt} = \frac{D_O\mathbf{v}}{Dt} + \frac{D_c\mathbf{v}}{Dt} + \frac{D_s\mathbf{v}}{Dt}, \tag{5.10}$$

$$\frac{Da}{Dt} = \frac{D_Oa}{Dt} + \frac{D_ca}{Dt} + \frac{D_sa}{Dt}. \tag{5.11}$$

While in Eq. 5.8 and Eq. 5.9 the outer process is ignored, it is included in Eq. 5.10 and Eq. 5.11, i.e., they define the overall dynamics on the surface.

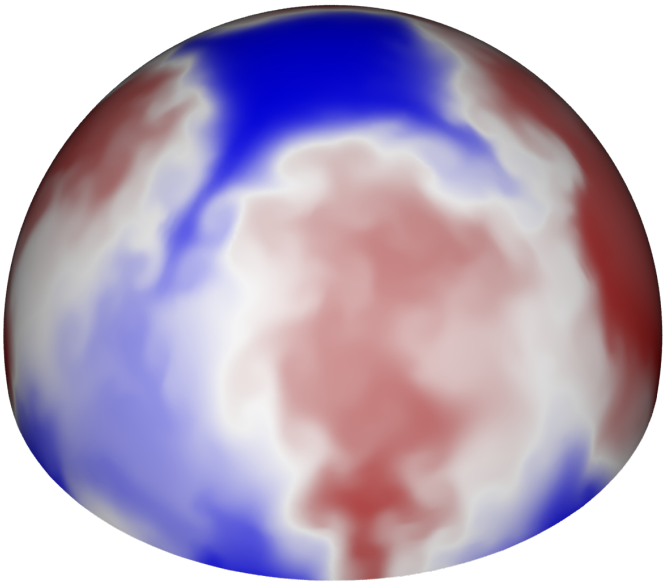


Figure 5.6: Thermal convection on a hemisphere. The temperature is color-coded, where the temperature rises from blue over white to red. Some areas are heated and others are cooled. Due to these temperature differences, buoyancy-driven flow arises.

5.2.5 Examples

To show the versatility of the model, various physical phenomena on the surface are modeled with different sets of PDEs.

Fluid Flow. Probably the most common application of the presented model is the simulation of fluid flow on an evolving surface as shown in Figs. 5.5, 5.7b, and 5.11. The example that accompanied the last section is the flow of an oil film on a water surface (Fig. 5.2). When the water surface moves due to the outer fluid simulation, due to friction forces, the oil fluid on the surface moves along. To model this kind of viscous fluid flow on the surface the Navier-Stokes momentum equation is used. It is written as

$$\frac{D_s \mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla_{T_\mathbf{p}M} p + \nu \nabla^2_{T_\mathbf{p}M}\mathbf{v} + \frac{1}{\rho}\mathbf{f}_b, \tag{5.12}$$

where p is the fluid's pressure, ρ is the density, ν is the viscosity, and $\mathbf{f}_b$ are body forces (e.g., gravity).

To describe the change of density, Eq. 5.11 is used. As density is an intensive property, the surface evolution part leads to

$$\frac{D_O \rho}{Dt} = -\rho(\nabla \cdot (\tilde{\mathbf{u}}_\mathbf{n})_{T_\mathbf{p}M}). \tag{5.13}$$

When modeling sinks and sources where the density in the outer process can be transferred to the inner simulation, the change due to the coupling part is

$$\frac{D_c \rho}{Dt} = -s_3 \rho^{\text{rel}}. \tag{5.14}$$

Finally, from the general continuity equation the third part for the change of density can be written as:

$$\frac{D_s \rho}{Dt} = \rho(\nabla_{T_{\mathbf{p}}M} \cdot \mathbf{v}), \tag{5.15}$$

where $\mathbf{v}$ is the fluid's velocity in the inner process. Putting the three parts together leads to

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot (\tilde{\mathbf{u}}_{\mathbf{n}})_{T_{\mathbf{p}}M}) - s_3 \rho^{\text{rel}} + \rho(\nabla_{T_{\mathbf{p}}M} \cdot \mathbf{v}), \tag{5.16}$$

Please note that if incompressible fluid flow for the inner process is modeled, i.e., this means $F_2(\rho, \mathbf{v}) = 0$, there may still be divergence from the outer process. Therefore it cannot be set to zero, but rather has to be set to

$$\nabla_{T_{\mathbf{p}}M} \cdot \mathbf{v} = \nabla \cdot (\tilde{\mathbf{u}}_{\mathbf{n}})_{T_{\mathbf{p}}M}, \tag{5.17}$$

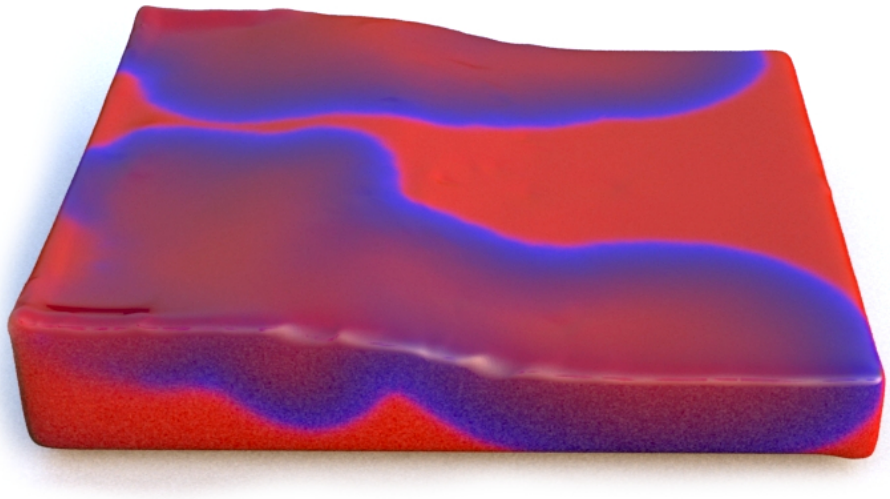
so the density changes come solely from the sinks and sources from Eq. 5.7.

Buoyancy-induced Flow. To demonstrate that the method is suitable to simulate natural convection on a spherical surface, an outer process was defined with a static hemisphere as shown in Fig. 5.6. In this example, the outer velocity was defined as $\mathbf{u} = 0$. This means the surface evolution part of our derivative, $\frac{D}{Dt}O$ is zero.

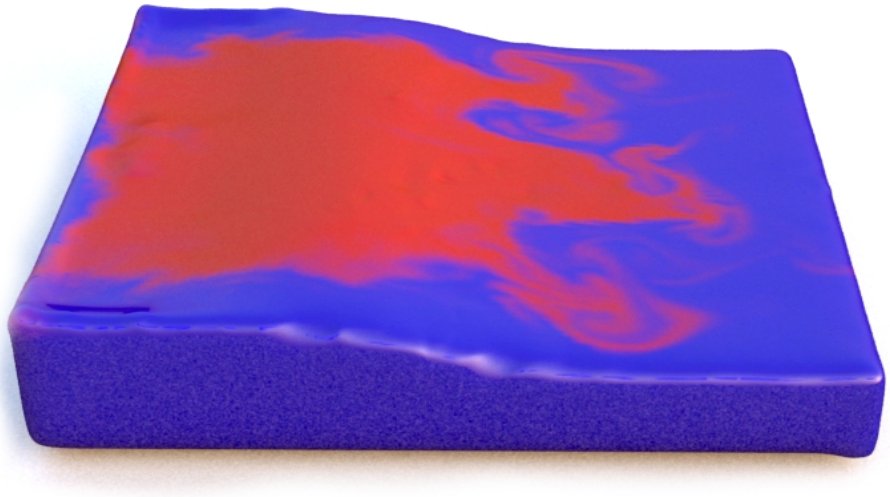
The coupling part is used for some areas on the surface that are heated and cooled by outer temperature constraints. Temperature transfer is modeled according to Eq. 5.7 and, therefore, the fluid on the surface changes its temperature.

Due to temperature differences in the fluid, natural convection arises, which is modeled in the part for $\frac{D_s \mathbf{v}}{Dt}$. To model the buoyancy-driven flow, the Navier-Stokes equation from above is extended with the Boussinesq approximation [19]. The temperature differences create density variations: $\rho = \rho_0 - \eta\rho_0(\mathcal{T} - \mathcal{T}_0)$, where η is the coefficient of thermal expansion, $\mathcal{T}_0$ the reference temperature, and ρ_0 the reference density. An ideal and incompressible gas is assumed and, therefore, $\eta = \frac{1}{\mathcal{T}_0}$. Furthermore, gravity is the only body-force in the model, therefore Eq. 5.12 becomes

$$\frac{D_s \mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla_{T_{\mathbf{p}}M} p + \nu \nabla^2_{T_{\mathbf{p}}M} \mathbf{v} + \left(1 - \frac{\mathcal{T}}{\mathcal{T}_0}\right)\mathbf{g}, \tag{5.18}$$



(a) Reaction-diffusion



(b) Fluid flow

Figure 5.7: This approach can model different behaviors on the same input data. The upper image shows a reaction-diffusion simulation on top of a dam break SPH simulation. The lower image shows a fluid flow on top of the same input simulation.

where $\mathbf{g}$ denotes the gravitational constant. The changes in the temperature field are modeled with the convection-diffusion equation:

$$\frac{D_s \mathcal{T}}{Dt} = \mu_d \nabla^2_{T_{\mathbf{p}}M} \mathcal{T}, \tag{5.19}$$

where μ_d is a constant diffusion coefficient.

Reaction-diffusion. To show PDEs that are outside the realm of Navier-Stokes, reaction-diffusion equations were modeled on the surface. These equations are commonly used to model chemical reactions of one or more substances (e.g., a substance is transformed into another due to chemical reactions) and the diffusion of the substance(s) in space. Fig. 5.7a shows such a process. Most reaction-diffusion terms in literature exist for two-component reaction-diffusion processes, their common structure is

$$\frac{D_s}{Dt} f_1 = R_1(f_1, f_2) + \mu_{d_1} \nabla^2_{T_\mathbf{p}M} f_1 \ , \tag{5.20}$$

$$\frac{D_s}{Dt} f_2 = R_2(f_1, f_2) + \mu_{d_2} \nabla^2_{T_\mathbf{p}M} f_2 . \tag{5.21}$$

The concentrations of the two chemicals are given with f_1 and f_2. The functions R_1 and R_2 are the reaction terms that characterize the system, μ_{d_1} and μ_{d_2} are the diffusion rates.

A typical set of reaction terms is the one proposed by Gray and Scott [37], which was used to produce the image in Fig. 5.7a:

$$R_1(f_1, f_2) = -f_1 {f_2}^2 + \beta(1 - f_1), \tag{5.22}$$

$$R_2(f_1, f_2) = f_1 {f_2}^2 - (\beta + \gamma) f_2 . \tag{5.23}$$

The chosen values for the constants of the feed rate β, the kill rate γ, and the speed coefficients μ_{d_1} and μ_{d_2} have a big impact on the course of the chemical reaction. The parameters to produce Fig.5.7a were set to $\beta = 0.03$, $\gamma = 0.06$, $\mu_{d_1} = 0.1$, and $\mu_{d_2} = 0.1$.

5.3 METHOD

This section describes how to apply the theory from Section 5.2 to create a simulation system. The method can be divided into seven steps, as illustrated in Fig. 5.2. The method starts with a coarse input (Fig. 5.2, Step 1), e.g., a fluid simulation. Then, it creates a signed distance field out of this simulation (Fig. 5.2, Step 2) and writes the simulation properties such as velocity, density, or color into 3D fields that are laid out in a narrow-band grid, which is the generic input format (Fig. 5.2, Step 3). Quantities from the outer process are brought into the surface domain in a coupling step and are compensated for the surface evolution as described in Section 5.3.4 (Fig. 5.2, Step 4 and 5). Then, the method simulates 2D details to enhance the coarse input by solving a PDE in the 2D surface space (Fig. 5.2, Step 6) that results in a velocity field that advects all grids in the last step (Fig. 5.2, Step 7).

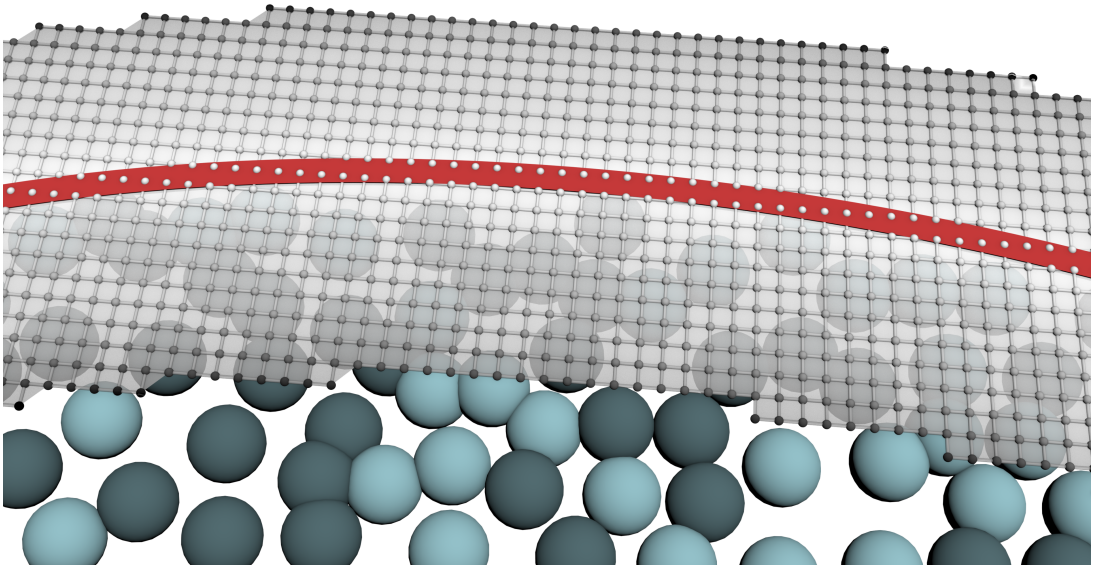


Figure 5.8: The distance field is stored in a narrow-band around the surface. In the same way, velocity and scalar fields are stored in a sparse data structure.

To solve a PDE on a surface the CPM is used [77] and is calculated in a 3D narrow-band. The main idea of the CPM is that if the values of a field are constant along the normal of the surface, many equations calculated in this 3D field are the same as if they were calculated in the tangent spaces of the surface. Section 5.3.3 describes the process to convert 3D fields into fields that have constant values along the surface normal.

The method is implemented in a way where each part is interchangeable using a system of modules that change data flowing through the system. The following describes each step in detail.

5.3.1 Data Input

The first step implements data input and can process a variety of data types. The examples include keyframe animation (Fig. 5.5), SPH particle simulation (Fig. 5.13 and 5.7a), and FLIP simulation (Fig. 5.11). The only requirement to be usable as an input is that the surface geometry data can be converted into a signed distance field and that values for the surface velocity can be generated on the surface.

5.3.2 Convert to Signed Distance Field

For converting polygon meshes to signed distance fields, several methods are available, e.g., [91], [102] and [53]. A special case arises when particle systems are converted to distance fields. Inside of the fluid, there are particles

everywhere. Here, the distance field obtained by these methods is not usable since the distance to the surface is needed and not to the nearest particle. For particle-based simulations, either the simulation can be meshed to a polygon mesh first or can be converted directly from particles to signed distance fields by defining an implicit surface from the particles [112] and then taking the distance to this implicit surface.

5.3.3 Create Narrow-Band Grid

A narrow-band around the surfaces is created to capture the velocity, density, and other properties of the flow from the nearest surface point. Based on the distance field, only cells near the surface are filled. Cells that a farther away than a certain threshold are left empty, as illustrated in Fig. 5.8. The algorithm keeps track of two velocity fields, the internal velocity field (denoted as $\mathbf{v}$ in Section 5.2) and an external velocity field (denoted as $\mathbf{u}$). It writes the velocity of the incoming surface into the external velocity grid. Depending on the use case, the method either initializes the internal velocity grid with a snapshot of the external velocity (for example, Fig. 5.13), or it creates a custom initialization routine to define the initial internal velocity, for example, by initializing with vanishing velocity (Fig. 5.5). The algorithm also keeps track of the scalar properties that are used in the simulation, e.g., the concentration of substances for the reaction-diffusion example.

To use the CPM the inputs must be narrow-band grids where the values are constant along the normal direction of the surface. Using an integer look-up grid that stores the cell coordinates of the closest point of the surface in each grid cell, it is possible to fill in a grid with values from the closest surface point. This step will be referred to as the CPM extension. Here, the values near the surface are extended along the normal direction, as can be seen in Fig. 5.9. The CPM extension is the final step in the narrow-band creation phase.

5.3.4 Surface Evolution

Before velocities or scalar values like color or density are used in a PDE, first the correction steps for density and velocity are executed, as described in Section 5.2.

For mass conservation, Eq. 5.3 has to be solved. As a building block for this calculation, a module is needed that implements the operator $\nabla \cdot (\tilde{\mathbf{u}}_{\mathbf{n}})_{T_{\mathbf{p}}M}$. This operator will calculate a scalar value for the divergence of the velocity field but with respect to the tangent space of a surface defined by the gradient of the input distance field. The result is then applied to the scalar fields that need correction. The interesting part of this divergence calculation is that the original 3D velocities are used for the divergence operator. The velocities

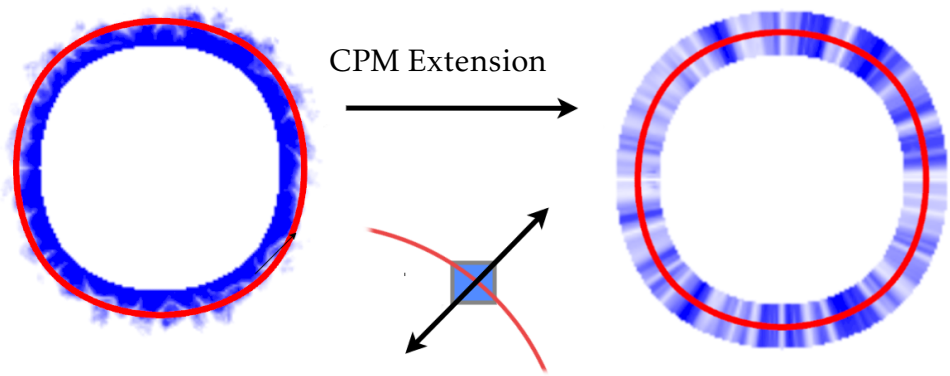


Figure 5.9: The CPM extension step extends the values closest to the surface along the normal direction. The red line indicates the surface. The blue color is a scalar value in the volume. On the left is the input field. On the right is the resulting CPM extension.

of adjacent cells need to be taken into account but projected to the surface normal of the current grid value, not the normals of the respective neighbor cells. An example to better understand the difference is the one of a growing sphere. Although the 2D velocities in surface space are zero at each point, the divergence is not. To get correct 2D divergence where a growing sphere causes sinks, and a shrinking sphere creates sources in the mass conservation equation, the velocities of neighbor cells have to be projected using the normal of the current cell.

For the conservation of momentum, the "Project Vector" module alters the velocities based on the surface tangent. Given a source vector grid and a distance field gradient, this module will project the input vectors onto the surface by subtracting the normal vector component from the gradient field. There is an option to maintain the length from the input vector, which resembles rotating the vector down onto the surface, as shown in Fig. 5.4. To apply Eq. 5.4, a module is added to calculate the curl of a velocity field and use it to rotate the velocities of a second velocity field. Both modules are applied to the velocity field for the internal velocity.

After the values are adapted based on the surface evolution, they are coupled to the values from the underlying simulation.

5.3.5 Coupling of Dynamics

For coupling the velocities, the velocity grids are sent to a module that applies Eq. 5.6 to the inner velocity. The parameters for $s1$ and $s2$ are exposed to the

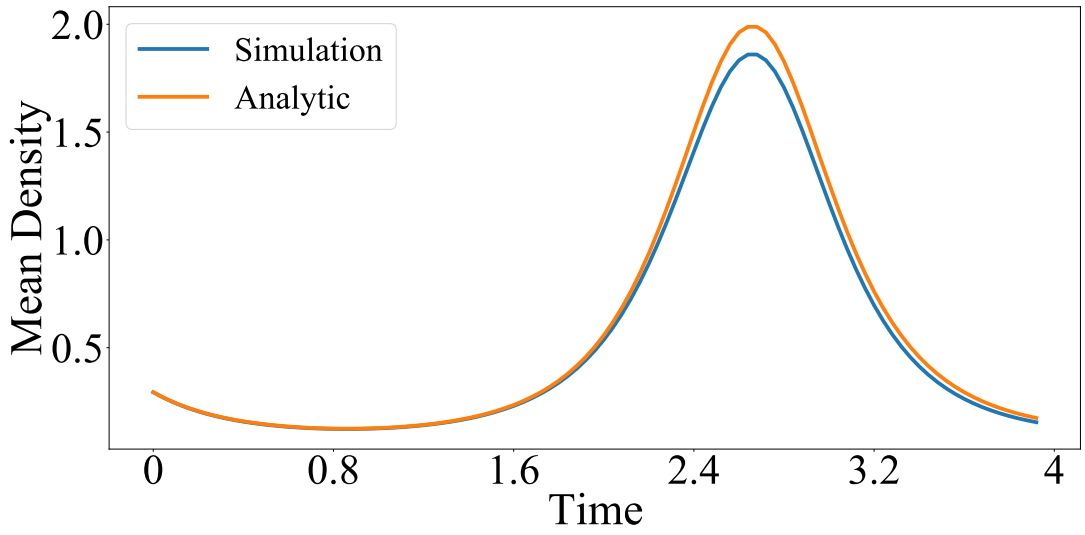


Figure 5.10: Simulated density over time on a growing and shrinking sphere compared to the analytic solution. The simulation is able to reproduce the analytical result. The numerical diffusion is neglectable.

user. Functionality to add noise, viscosity, and vorticity to the inner velocity is implemented. These parameters allow one to improve the coupling with additional details, as can be seen in Fig. 5.5.

By coupling scalar values from the initial 3D simulation to the 2D space using Eq. 5.7, mass transfer can be modeled. The parameter $s3$ is exposed to the user. It drives how strongly the outer process influences the values in the 2D simulation. A value of 0 would only initialize the 2D simulation, and from then on, the 2D simulation would be independent.

5.3.6 Solve PDE and Advect

As mentioned, the method operates on 3D grids where values do not change along the normal direction. The CPM allows solving PDEs in the surface space using these grids. Different sets of PDEs are implemented that can all be solved using the new modules. The modules work in 3D, but they were specifically designed for the fields that result from the CPM extension. Due to the CPM, the gradient naturally operates in surface direction, but the divergence, curl, and Jacobian have to be changed: When applying them, the projected version is used as described in Section 5.2, i.e., the input vector is projected onto the surface.

To simulate fluid flow on the surface, Eq. 5.12 is solved. Here, the projected divergence operator $\nabla \cdot (\tilde{\mathbf{u}}_\mathbf{n})_{T_\mathbf{p}M}$ is used. The "Divergence-Free" module will remove divergence with respect to the tangent space of a surface and take the di-

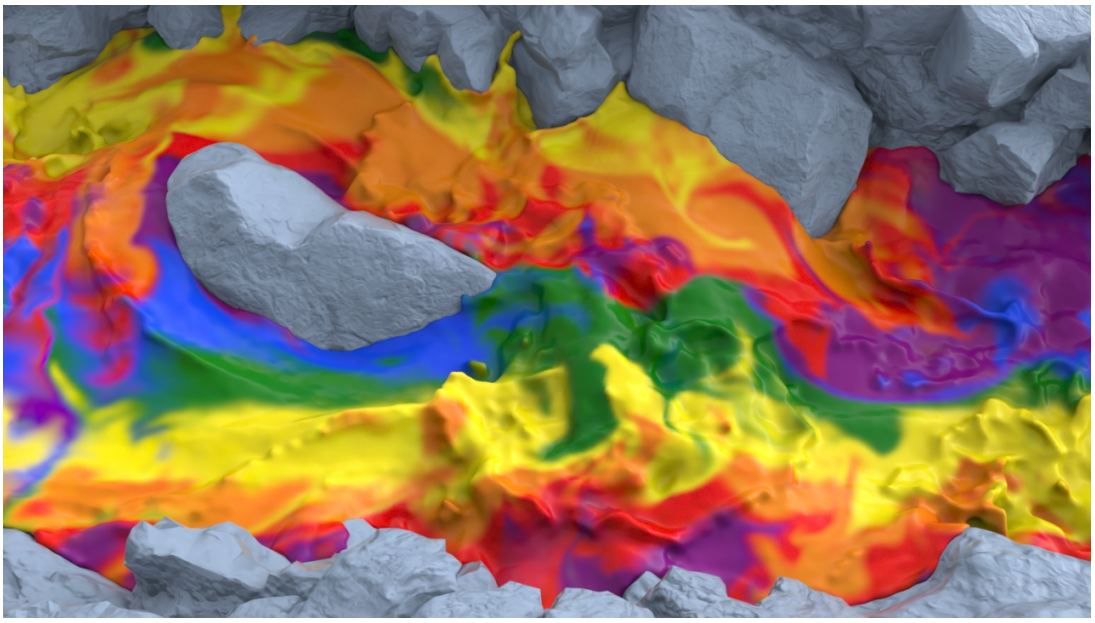


Figure 5.11: Fluid flow simulation on top of a river bed simulation. The decoupling of the simulation on the surface from the base simulation allows for interesting effects.

vergence of the surface evolution into account. For the Buoyancy-induced flow, the gravity is scaled depending on temperature as described in Eq. 5.18 and a diffusion module for temperature diffusion is added. To simulate reaction-diffusion for the example shown in Fig. 5.7a a module to implement Eq. 5.20 and Eq. 5.22 was written.

As the last step, all fields are advected using Eq. 5.10, i.e., with the resulting velocities.

5.3.7 Implementation

The modular approach can be easily implemented with existing frameworks like the SideFX™Houdini software package, which is widely used for VFX creation and offers an integration of the OpenVDB libraries [69] as a sparse volume representation for the calculations. The implementation for Houdini is available as an open source library, which makes it easy to integrate the method into typical production workflows. The algorithm is separated into independent modules which are implemented as separate Houdini operators, to be used in Houdini's simulation node graph framework. The Houdini surface operator (SOP) network level is used to implement the algorithms on top of the OpenVDB volume data type. The OpenVDB framework offers a data structure, the OpenVDB grid, to create such narrow-bands. All subsequent calculations

employ this structure and only spend computation time where needed. The "CPM Extension" module is the central building block to build CPM solutions in Houdini. Here, nearest-neighbor interpolation is implemented as suggested by Kim et al. [51], but also box and quadratic interpolation are implemented. To generate the SPH base simulations, divergence-free SPH [16] with consistent Shepard interpolation [76] was used. When simulating fluid flow, vorticity confinement as presented by Fedkiw et al. [26] was used.

5.4 RESULTS

To show the versatility of the method, different physical phenomena were modeled as described in Section 5.2.5 and were tested on a variety of scenarios. Outer processes with different characteristics were chosen, from static (Fig. 5.6) to hand-animated meshes (Fig. 5.5), from coarse scale SPH-based fluid simulations, including sudden changes in topology (Fig. 5.7) to high-resolution multi-phase SPH simulations (Fig. 5.13). In all of the above situations, the method is able to add a fine-scale secondary simulation on top of these surfaces, revealing fine-scale details as promised.

5.4.1 Versatility and Simulation Quality

The approach is able to add effects onto the surface of the simulation. Simulating a second fluid flow on the surface enables one to add other phenomena on top of an existing simulation. As shown in the oil film example (Fig. 5.13), the effect of oil spreading on the surface is achieved by simulating a second, fine-scale fluid flow on the surface, which is just added to the base-simulation. As mentioned, the level of detail of the secondary simulation can be chosen independently, and increasing the simulation resolution of the underlying SPH simulation would not have the same effect. Instead, there would just be a scalar field with higher resolution, like the one shown in Fig. 5.13b and not having simulated the laws of the secondary flow. In addition, due to the modeled mass transfer, the oil particles that emerge from below the surface can contribute to the 2D simulation. This coupling introduces significantly more details than just a plain simulation on the surface itself. The amount of detail added is significant even for low-resolution base simulations, and it can be further improved (Fig. 5.13c to 5.13d) by increasing the resolution. Fig. 5.7a presents a reaction-diffusion of two chemicals simulated on top of a coarse dam break simulation. This example illustrates that the method can simulate not only a second flow equation on the surface but any kind of desired physical phenomena that can be described by a set of PDEs.

In the buoyancy-induced flow example (Fig. 5.6), thermal convection is simulated to demonstrate the emergence of isolated vortices on a static surface,

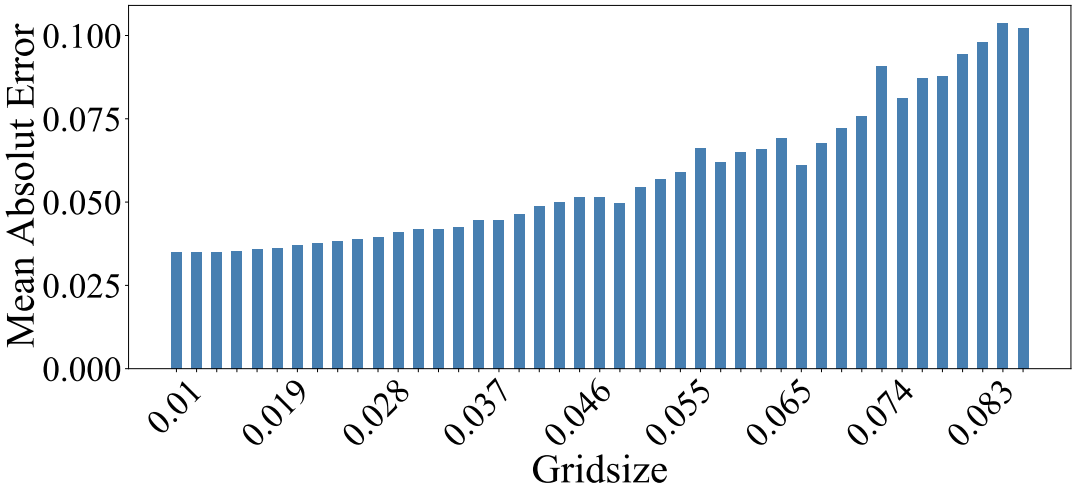
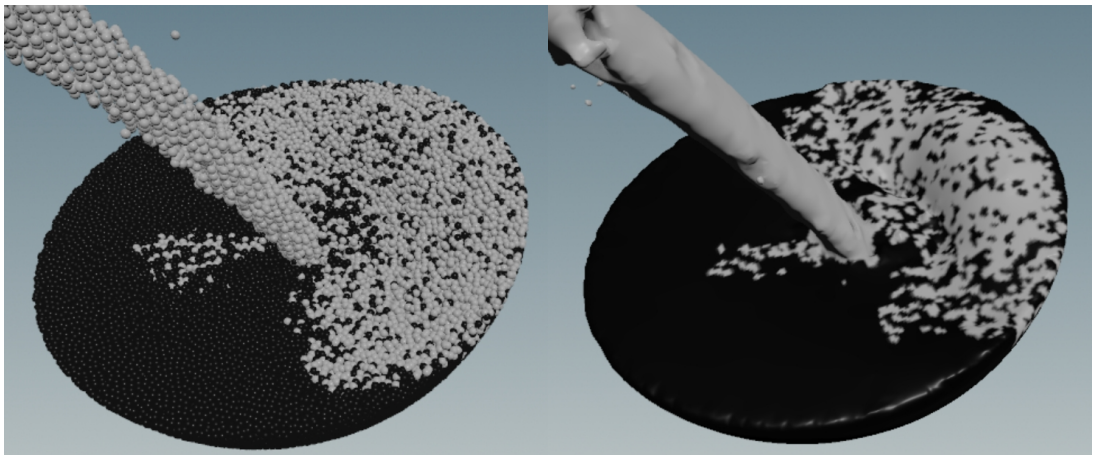


Figure 5.12: The mean error for different grid sizes. The smaller the grid cells, the closer the values are to the analytic solution.

replicating the physical experimental setup by Seychelles et al. [80]. The setup consists of a static hemisphere on a heating plate giving rise to thermal convection. The method can create small-scale vortex structures using the Boussinesq approximation. This example shows that the approach can model real-world alike physical phenomena.

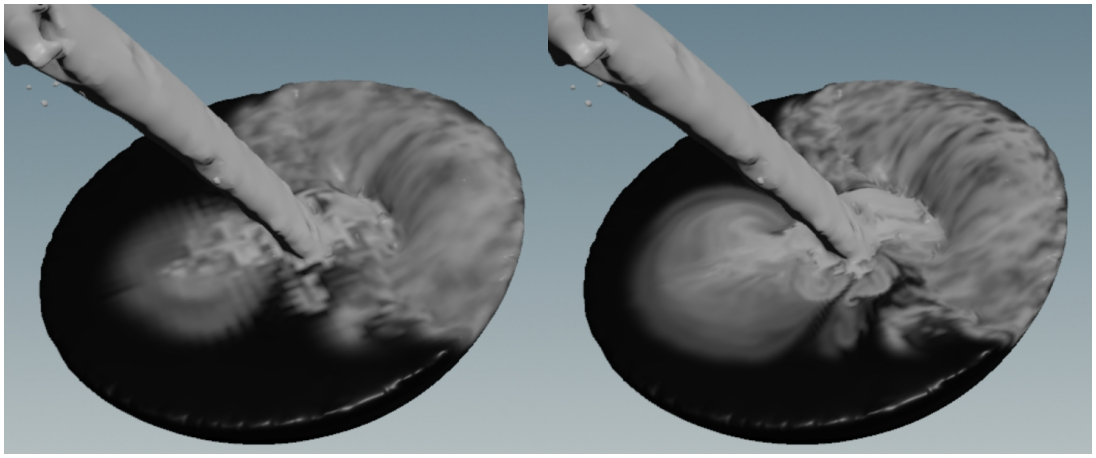
A hand-animated sphere with a periodically oscillating radius was used to test the approach on mass conservation. The mean density per surfactant was calculated and plotted as a function over time as a graph (Fig. 5.10). In this example, it is possible to analytically calculate the density change that is needed to ensure mass conservation and compare it with the presented simulated result. As shown in Fig. 5.10, it is possible to reproduce the analytical solution over time. The small loss in density can be attributed to numerical diffusion and is neglectable in this case. To test the accuracy depending on the grid resolution, this scenario was simulated with different grid resolutions. As can be seen from Fig. 5.12, the mass conservation approach works as expected and the simulation converges to the analytical solution using finer grid resolutions.

The author refers the reader to Appendix B for some still images of the animations and the accompanying video in the video appendix [63] to see some of the examples in motion. Moreover, a precompiled version of the plugin for Houdini is provided with two sample scenes for testing on Zenodo [64].



(a) Source SPH particles

(b) Original scalar field derived from particle attributes



(c) 2D simulation with grid resolution of 0.05 (d) 2D simulation with grid resolution of 0.02

Figure 5.13: Pouring polluted water into a bowl. The fine-grained simulation enhances the underlying SPH simulation. The presented mass transfer even captures oil particles that emerge from under the surface. Different resolutions can be generated independent from the input resolution. This allows iterative refinement of parameters as needed in typical VFX production workflows.

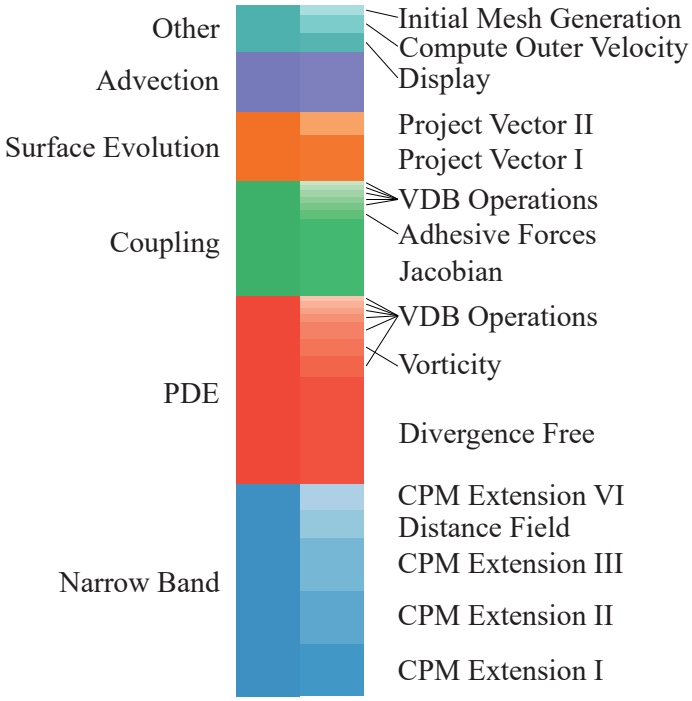


Figure 5.14: The calculation costs of the used nodes for the CPM methods are in the magnitude of the existing Houdini operators typically used in simulations.

5.4.2 Performance

The performance was measured for every operator individually with the rotating sphere example (Fig. 5.5). The grid resolution has the most significant impact on the computation time, but also the width of the narrow-band profoundly influences the performance. As can be seen in Fig. 5.14, the largest computation time for a single task is spent in the "Divergence Free" operator. This node uses OpenVDB's Poisson solver with a maximum of 100 iterations to obtain a divergence-free vector field. The second most computation time is spent to compute the Jacobian for the coupling step. In third place is Houdini's advection operator. All other operators have smaller computation costs. This shows that the computation time for the needed steps is in the magnitude of Houdini's advection node.

In contrast to the very special solution for the meniscus effect in Chapter 4, this chapter presented a versatile method to model small-scale effects that can be expressed with PDEs on an evolving surface.

CHAPTER 6

Conclusion

This thesis presented methods to add fine detail to fluid animation in the domain of visual effects to help achieve greater realism in CGI.

The overall goal of this work – to answer the question "What are effective strategies to augment large-scale fluid simulations with small-scale physical effects?" – was achieved in 3 steps. First, the naive brute-force approach was explored, then a smart analytical method for a particular effect was discussed, and finally, a more generic approach that can achieve a variety of effects was proposed.

6.1 Summary

Chapter 1 derived the motivation and stated the main research questions of this work. In Chapter 2, the meniscus effect and iridescence effects as examples of small-scale effects were described. Then, the Knudsen number of those effects of interest was determined to find an adequate physical model. With that, the Navier-Stokes equations were identified as the governing equations and briefly described. Then, the difference between the Lagrangian and Eulerian views was introduced and effective numerical methods for both approaches as they are used in later chapters were presented.

In Chapter 3, an approach was presented for simulating and rendering physical-based fluid effects in a VFX production environment using a client/server architecture that is practical for distributed simulation resources. The fluid simulation implemented smoothed particle hydrodynamics (SPH). The proposed

strategies of using GPUs for computation and the usage of cloud computing resources were tested and presented in a complete system architecture. The integration into commercial software packages was demonstrated. This system was developed to integrate into the particle system of Autodesk 3ds Max. It included a simulation component and a raytracing component for isosurface raytracing. The simulation component allowed outsourcing of the computational work to an external computing resource while displaying the results in an interactive session. A test setup was created where the computational work was executed on a cloud-computing resource from Amazon Web Services (AWS) and the interactive session was running on a desktop PC. The feasibility of the system was validated with a user study with industry experts.

The concept of surface particles was extended by introducing blind particles that facilitate efficient direct raytracing of isosurfaces. This also reduced the memory footprint of network transfer to create savings in both rendering time and storage requirements. The performance of the approach was evaluated with local and remote simulation on CPUs and GPU. The raytracing component was developed to satisfy the requirements of a production pipeline for visual effects. The rendering output included all necessary extra render elements like normal passes, velocity passes, specular passes as shown in Fig. 3.9.

In Chapter 4, an analytical closed-form solution for the meniscus shape was introduced, and then it was shown how to apply this to an SPH-based fluid simulation. Instead of simulating the meniscus effect with a full fluid simulation, the new approach uses a closed-form of the meniscus shape of the fluid interface to add a small-scale curved surface along the contact line at render time. This chapter's technical contribution is to use the closed-form fluid meniscus in a direct implicit raytracing approach to remove the limitations of mesh-based meniscus modeling. The method uses the analytical solution of fluid menisci to enhance fluid surfaces based on physical simulations. The author presented a new approach for achieving physically correct contact angles at fluid–solid boundaries in the surface generation stage. The method blends the correct contact angle inclination into the original surface shape in a physically plausible way. For the rendering step, a raytracing routine for the commercial renderer VRay was implemented. Several test scenes were created to test the usability of the approach.

Chapter 5 presented a method to solve PDEs on evolving surfaces. In contrast to Chapter 4, where only one particular effect was discussed, this last chapter presented a method that enables a variety of effects near the surface that can be described with a PDE. A method to solve PDEs on evolving surfaces was presented where the external movement was considered as the driving process. Physically motivated conservation laws were derived and coupling strategies were presented. Simulations of 3D and 2D space were coupled in a way that allows computing high-resolution 2D simulations on coarse

input surfaces efficiently and that exposes physical plausible parameters to control the strength of the coupling. The examples demonstrated that the approach can be used for different types of problems that require solving PDEs on a surface. It was shown how to integrate all necessary steps into a VFX production environment in a modular way so the building blocks can be reused and a reference implementation was provided.

6.2 DISCUSSION

This work presented three different strategies on how to augment fluid simulations with small-scale physical effects to properly address the **Overall Research Goal**, "What are effective strategies to augment fluid animations with small-scale physical effects?" The first strategy was to compute the solution with a brute-force approach where the resolution is increased as presented in Chapter 3. The second strategy was to find an analytical solution to the problem and combine this with a coarser simulation as described in Chapter 4. The third strategy was to limit the computation to a subset of the initial domain to save effort as presented in Chapter 5, where the simulation of the small-scale effect was computed in the 2D surface space instead of the full 3D space.

Each of these strategies has its reason to exist. To choose the right one, the advantages and disadvantages should be known. **Research Question 1**, "What are strategies to improve existing methods to such an extent that the resolution can be increased enough to achieve small-scale effects?", explored the brute-force method. This method seems unattractive at first glance as the increased resolution creates problems at various places. The obvious points are the increased amount of simulation data and smaller time steps that both lead to increased computation time for the simulation. More simulation data means not only an increased RAM requirement during the simulation but also a greater need for disk storage for the simulation results. If the simulation is computed on cloud resources, the results often have to be transferred from cloud servers to on-premise machines, which leads to higher bandwidth requirements and transfer costs. The high amount of simulation data will often also increase RAM requirements for the renderings stage and increase render times. The increased computation time may prolong the iteration cycle of shot production for the artists.

Despite these disadvantages, the brute-force method should still be considered as a possible strategy, as there are some advantages to the approach. It is important to consider all of the efforts involved in a production. Often computation time is cheaper than artist and developer time. If existing tools are used, the artists are already trained to use them and the correctness of their simulation has been tested in production. But if instead of simply increasing the resolution

of an existing toolset, a new solution needs to be implemented, that solution needs to be tested and taught to the artists as well. This additional effort may not be justified, especially if a certain effect is only needed once. According to Zia Ullah et al. [113], cloud computation costs are halved every three years and the computation power is rising constantly following Moore's Law [38]. With these advantages, it is always an option to consider the brute-force strategy.

The presented blind particle approach reduced the maximum memory usage and transfer costs for a particle cache and helped to reduce render time for raytracing. In the context of a VFX production pipeline as described in Chapter 2, the approach presented in Chapter 3 adds small-scale effects in the "Simulation" step of the pipeline, although some improvements were added in the "Rendering" step with the use of direct raytracing.

The second strategy to add small-scale effects was to use analytical methods in combination with a simulation. The typical curved meniscus shape that water forms at the borders produces characteristic highlights and caustics that can be seen from far away. Therefore, adding a curved meniscus shape in large-scale simulation at contact lines can greatly improve the visual quality. The idea of Chapter 4 is to add this meniscus shape with the help of an analytical solution without the need for fine-grained simulation. With this approach, the effect can be decoupled from the simulation stage. The simple control with one parameter, and the low computational footprint make this approach especially applicable to visual effects pipelines that need a fast turnaround time for the daily work or any kind of real-time visual simulation.

Using the example of the meniscus effect, Chapter 4 answered **Research Question 2** of how analytical solutions and simulation results can be combined to achieve small-scale effects effectively.

The presented surfacing method can be applied to existing fluid simulators without any change to the simulation itself. The solution can be applied to a wide range of simulation methods as it operates on the distance fields of the fluid simulation and the collision objects. In the context of a VFX production pipeline, this method does not operate in the "Simulation" step, instead, the small-scale meniscus effect is added in the "Rendering" step. The surface is generated locally at the ray–surface intersection step of the raytracing algorithm without generating triangles. Therefore, the characteristic highlights that are created by the typical curved meniscus shape that water forms at the borders can be captured by raytracing systems even when they are on a sub-pixel scale. Direct raytracing always creates the needed resolution on a sub-pixel level and solves the problem that otherwise a very dense polygon mesh for the surface would be needed to capture the small meniscus curve. This dense polygon mesh would need a lot of additional storage for saving the scene and a lot more memory for rendering. Heavy polygon meshes would slow

down not only the rendering but also overall scene handling for loading/saving and navigating the 3D scenes.

The method introduces only little computational overhead to existing surfacing methods while significantly improving the visual quality even for simulations with high water volumes from SPH or level set simulations. In certain situations, such as a meniscus between a horizontal fluid and a straight wall, the proposed method is independent of the simulation resolution and identical to the exact solution.

One limitation to this approach is that the physical simulation is not influenced by the capillary effects. Various effects can therefore not be reproduced like, e.g., the capillary rise in a cylinder. To produce these effects methods that use the capillary forces in the simulation itself have to be employed.

Another disadvantage of this method is that other effects that require feedback for the simulation or scene setup phase cannot be implemented as the meniscus does not exist in these phases. To compensate for this weakness, a plugin for the Autodesk 3ds Max was written that provided a polygon-based preview of the meniscus effect in an interactive session.

But this specialized solution is applicable only for this one particular effect, the meniscus effect, not for other types of effects. This leads to the main disadvantage of the second strategy: It is very hard to find effects that are suitable for an analytic approach. Most fluid scenarios are too complex and require PDEs. Sometimes effects can be isolated and repeated at different positions. For example, the circular wave patterns around raindrops that fall into a calm water surface can be computed with an analytic solution and then be added for every raindrop. A slight variation of the idea of analytic solutions is to use precomputed solutions for an isolated area and apply this solution at multiple locations. For example, Garg et al. [32] built a database for image-based rain streak rendering by precomputing the variations in streak appearance with respect to lighting and viewing directions. These small-scale lighting effects inside of a falling raindrop are then applied to a large rain scene. If there is no possible analytic approach or if the effect cannot be isolated and reapplied, the second strategy is not applicable.

This leads to **Research Question 3** on how fine-scaled effects near the surface can be added to fluid simulations more generically. Essentially, the method presented in Chapter 5 is a trade-off between physical correctness and visual resolution. The presented method couples simulations of 3D and 2D space to add fine-scaled 2D simulations on coarse input surfaces. The simulation operates on a sparse volume data structure that tracks a narrow band near the surface. With this approach, the additional computation time that is needed to simulate the desired surface effects is only spent on a small band of cells around the surface. This is more efficient than simulation on a full 3D

grid of the whole fluid body. One could say, the method trades the physical correctness of a true 3D simulation with the visual richness of a high-resolution 2D simulation on the surface. In many VFX scenarios, high-resolution effects with not entirely correct but plausible movement can be more believable than correct simulations that do not have enough resolution. The 2D surface simulation can add a big visual difference to scenes.

The weak point of the third strategy is that it is hard to create physically correct models. The reduction of the simulation space to the two-dimensional space creates constraints on how material can move. Even for effects that are supposed to happen mostly on the surface, e.g., an oil film on a water surface, the correct model is a full 3D simulation. The 2D approach is just an approximation to create visually similar results. Although the method is physically not correct, physically motivated conservation laws were derived and coupling strategies were presented, so plausible effects can be achieved.

When deciding on one of the strategies, one factor is the working time that is required to create the desired number of shots. This includes not only the time it takes to work on the actual effects but also training time for new tools for the artists. Therefore, it is essential to integrate new methods as tightly as possible into existing workflows and tools. In this work, for each approach, it was shown how to integrate all necessary steps into a VFX production environment. For Chapter 3 and Chapter 4, the methods were integrated into the plugin system of Autodesk 3ds Max, a content creation software package that is widely used in the VFX industry. In the later period of this research work, the software package Houdini became more accessible to researchers with a free personal learning edition. To seize this opportunity and get a broader picture of the tools used in the industry, for Chapter 5 a plugin for Houdini was written that provided functions as modular nodes for the software's node-graph system. Several test scenes were created in each of the systems to verify if the approach can be applied in a VFX production environment. The lesson learned from implementing for those two different packages is that it is generally better to break the functionality down into the smallest possible modules. The node-based approach in Houdini was easier to implement and test. As a user, it is easier to learn how to use the individual components and you can find creative ways to use the functionality.

6.3 FUTURE RESEARCH DIRECTIONS

For the brute-force approach in Chapter 3, only remote execution of the simulation on a single GPU was demonstrated. The simulation workload was outsourced as a whole and not divided into chunks. However, by introducing a spatial division of the particles as described by Ihmsen et al. [44], the

implementation could be extended to a multi-GPU system that runs on many cores in parallel to be employed in scalable hardware environments such as GPU clusters on the server-side and make small-scale effects possible on larger simulation domains. Distributing simulations to many cores is not a new topic, but with the shown approach of invoking this from an interactive session and using cloud resources, there are some new ideas possible. Additional cloud GPUs could be added to the simulation on-demand based on heuristics that take current pricing and GPU availability into account. Most cloud vendors offer computing instances with variable pricing based on current demand, e.g., the spot-instance feature from AWS. With the information on how many GPUs are available in the cloud for simulation and the current pricing, it is possible to estimate the time and costs for an effect. This information would then be relevant for the decision if a brute-force approach for this small-scale effect makes sense in this case.

If the distributed approach could be combined with adaptively sampled fluid simulations as described, e.g., by Adams et al. [4], the method would become the ideal solution to add small-scale effects to simulations. This would add detail only where it is needed and would be similar to the third strategy of limiting the simulation to the 2D space of the surface, but without losing physical correctness. The generic client/server architecture could be extended to other fields of physically based simulation. Similarly, other commercial 3D packages could be integrated with the system.

Another possible field for future work is to combine the results from Chapter 4 and Chapter 5. The analytic closed-form solution could be integrated into the sparse narrow-band data structure of Chapter 5. Then the results could easily be fed into other functions or operators, e.g., to also include an oil film on the surface that has a meniscus shape. This idea would already require implementing different boundary conditions for obstacles into the method from Chapter 5 as this assumed a free flow on the surface without obstacles. Implementing different boundary conditions for obstacles would therefore be a mandatory next step.

Testing further types of PDEs on moving surfaces is another topic for future work. This work investigated the iridescence effect of an oil film on water. There are more small-scale effects with water that are often relevant in VFX productions. One area is the domain of wetting effects when surfaces change their appearance once they get in contact with water. Often wet maps are used to create this effect. Simulating the diffusion of water inside of media like cloth was shown by Huber et al. [41]. This is not a water surface effect in the strict sense, but the idea of solving the water movement in 2D space with the approach of Chapter 5 should be applicable. This might also be true for thin streams and droplets of water on surfaces. This could lead to efficient simulation and rendering when used in combination with the surface

rendering of Chapter 4, where a rendering of a drop of water with a correct contact angle from a single SPH particle was shown.

This work aimed to establish a research area for the addition of small-scale effects to fluid animation in CGI to increase the plausibility of an effect. It presented examples of such effects and contributed practical methods that can be implemented. The presented methods showed that it is possible to decouple the small-scale effect from the base fluid simulation while still obtaining convincing visual results. All of the methods investigated in this work employ standard physically based methods. However, with the methods presented, physical correctness was sacrificed in favor of computation time. The most important factor is whether the effect is perceived as credible by the viewer. This naturally leads to the field of human perception of effects and the ingredients required to make an effect more convincing. If the visual ingredients for small-scale effects that improve the plausibility of CGI could be classified, then this might be a good application for artificial intelligence (AI) with generative adversarial networks [100], where the needed features can be learned with machine learning (ML) methods and applied to raw renderings. With such methods, maybe a fluid meniscus could be added to renderings without any knowledge about the fluid but solely based on the rendered image.

APPENDIX A

System Survey Results

Supplemental material for Subsection 3.5.2.

Here, the results of the quantitative part of the user study are detailed. The first part consists of the specific questions related to our implementation, the second part shows the System Usability Scale (SUS) questions.

As stated in the text, it can be seen that all participants agreed that the overall workflow is better if the fluid simulation is integrated in the 3D content creation software in contrast to standalone fluid simulation applications.

Only one user agreed to the statement "Outsourcing simulation calculation to a render slave speeds up my workflow". The reason for this became clear in a discussion with the participants. While all participants had the latest GPUs in their workstations, none of the studios had GPU blades in their server farms yet. The user who agreed to the statement referred to a possible future situation where GPU servers would be available, while the other users referred to their current situation where the fastest GPUs were only available in the workstations.

The usability was also rated very good in the SUS questionnaire. This is attributable to the fact that the solver blended into the interface they were experts in.

A.1 SYSTEM USABILITY SCALE

1. I think that I would like to use this system frequently

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	33,3% (1)	**66,7% (2)**	4,67	3

2. I found the system unnecessarily complex

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
100,0% (3)	0,0% (0)	0,0% (0)	0,0% (0)	0,0% (0)	1,00	3

3. I thought the system was easy to use

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	0,0% (0)	**100,0% (3)**	5,00	3

4. I think that I would need the support of a technical person to be able to use this system

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
100,0% (3)	0,0% (0)	0,0% (0)	0,0% (0)	0,0% (0)	1,00	3

5. I found the various functions in this system were well integrated

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	33,3% (1)	**66,7% (2)**	4,67	3

6. I thought there was too much inconsistency in this system

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
66,7% (2)	33,3% (1)	0,0% (0)	0,0% (0)	0,0% (0)	1,33	3

7. I would imagine that most people would learn to use this system very quickly

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	0,0% (0)	**100,0% (3)**	5,00	3

8. I found the system very cumbersome to use

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
100,0% (3)	0,0% (0)	0,0% (0)	0,0% (0)	0,0% (0)	1,00	3

9. I felt very confident using the system

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	33,3% (1)	**66,7% (2)**	4,67	3

10. I needed to learn a lot of things before I could get going with this system

					AnsweredQuestion	**3**
					SkippedQuestion	**0**
Strongly disagree				Strongly agree	RatingAverage	RatingCount
66,7% (2)	33,3% (1)	0,0% (0)	0,0% (0)	0,0% (0)	1,33	3

A.1.1 Additional Questions

1. Outsourcing simulation calculation to a render slave speeds up my workflow

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
66,7% (2)	0,0% (0)	33,3% (1)	0,0% (0)	0,0% (0)	1,67	3

2. The overall workflow is better if the fluid simulation is integrated in the 3D content creation software in contrast to standalone fluid simulation applications.

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
66,7% (2)	33,3% (1)	0,0% (0)	0,0% (0)	0,0% (0)	1,33	3

3. In 3ds Max the overall workflow is better if the fluid simulation is integrated in Particle Flow in contrast to fluid object plugins.

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
33,3% (1)	**33,3% (1)**	**33,3% (1)**	0,0% (0)	0,0% (0)	2,00	3

4. Integrating fluid solver in Particle Flow allows me to achieve a greater variety of effects than a standalone solver.

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
66,7% (2)	0,0% (0)	0,0% (0)	33,3% (1)	0,0% (0)	2,00	3

5. Direct raytracing of fluid blobs results in better visual quality

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
66,7% (2)	33,3% (1)	0,0% (0)	0,0% (0)	0,0% (0)	1,33	3

6. Direct raytracing of fluid blobs is less flexible

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	33,3% (1)	0,0% (0)	**66,7% (2)**	4,33	3

7. Direct raytracing is slow

				AnsweredQuestion		3
				SkippedQuestion		0
totally agree				totally disagree	RatingAverage	RatingCount
0,0% (0)	0,0% (0)	0,0% (0)	33,3% (1)	**66,7% (2)**	4,67	3

A.1.2 Background

1. Years of professional experience with 3D Software

		AnsweredQuestion		**3**
		SkippedQuestion		**0**
	ResponseAverage	ResponseTotal	ResponseCount	
Years	**8,67**	**26**		3

2. Years of professional experience in VFX.

		AnsweredQuestion		**3**
		SkippedQuestion		**0**
	ResponseAverage	ResponseTotal	ResponseCount	
Years	**3,67**	**11**		3

3. Experience with 3ds Max. Your daily usage in hours.

		AnsweredQuestion		**3**
		SkippedQuestion		**0**
	ResponseAverage	ResponseTotal	ResponseCount	
hours	**8,33**	**25**		3

4. Experience with VFX creation

			AnsweredQuestion			**3**
			SkippedQuestion			**0**
beginner			professional	RatingAverage		RatingCount
0,0% (0)	33,3% (1)	0,0% (0)	**66,7% (2)**	3,33		3

5. Experience with fluid effects. What percentage of your work is fluid related?

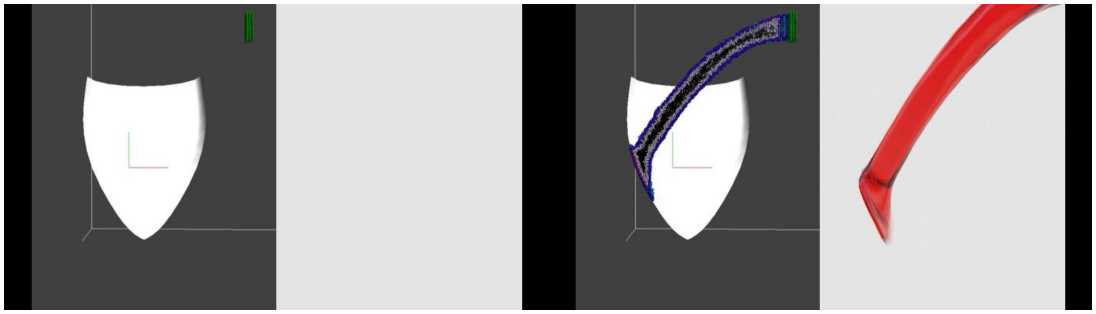
		AnsweredQuestion		**3**
		SkippedQuestion		**0**
	ResponseAverage	ResponseTotal	ResponseCount	
%	**48,33**	**145**		3

6. Experience with V-Ray renderengine. What percentage of your work is rendered with V-Ray?

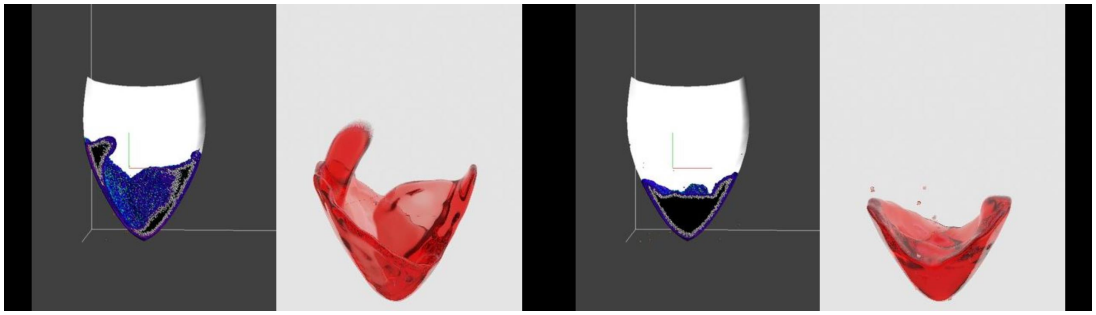
		AnsweredQuestion		**3**
		SkippedQuestion		**0**
	ResponseAverage	ResponseTotal	ResponseCount	
%	**99,67**	**299**		3

APPENDIX B

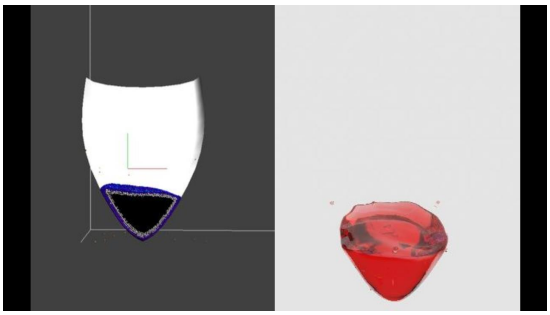
Samples



(a) t=0 (b) t=1s

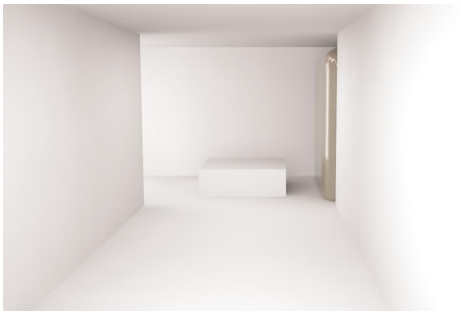


(c) t=2s (d) t=3s

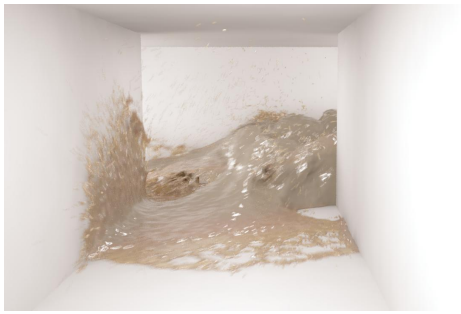


(e) t=4s

Figure B.1: Animation of fluid poured into a glass. On the left, SPH particles color-coded by particle type. Black: particles to be omitted, white: blind particles, colored: surface particles color-coded by pressure. On the right, the resulting rendering with raytraced motion blur. Frames in 1 second steps.



(a) t=0



(b) t=1s

(c) t=2s

(d) t=3s

(e) t=4s

Figure B.2: Dam break simulation with 1 million particles with direct raytracing of an isosurface with motion blur and global illumination. Frames in 1 second steps.

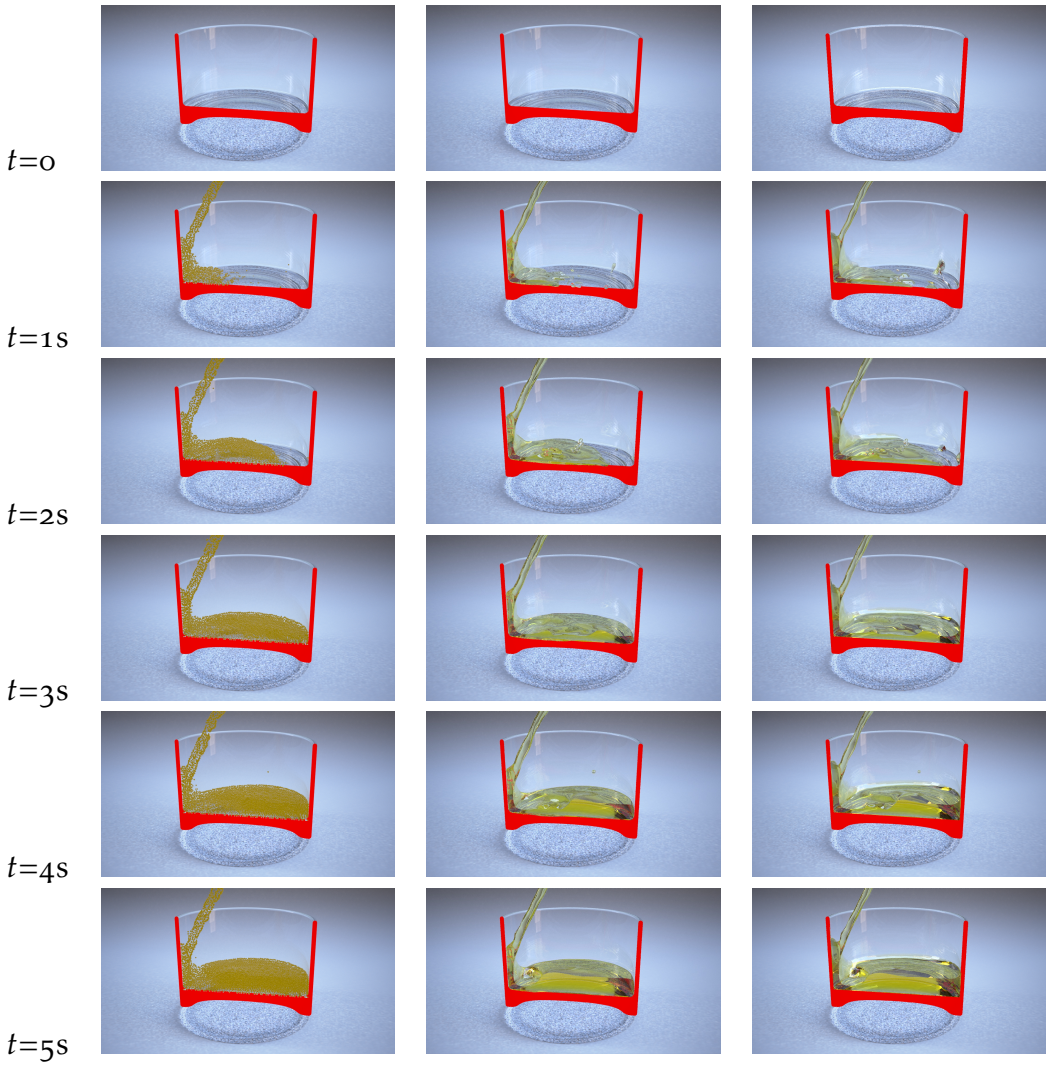


Table B.1: A fluid meniscus in a simulation of a fluid poured into a glass. The first column shows the underlying particle simulation. The second column depicts the resulting surface rendering without correction. The third column shows the corrected version with a fluid meniscus added at render time. Frames are in 1 second steps.

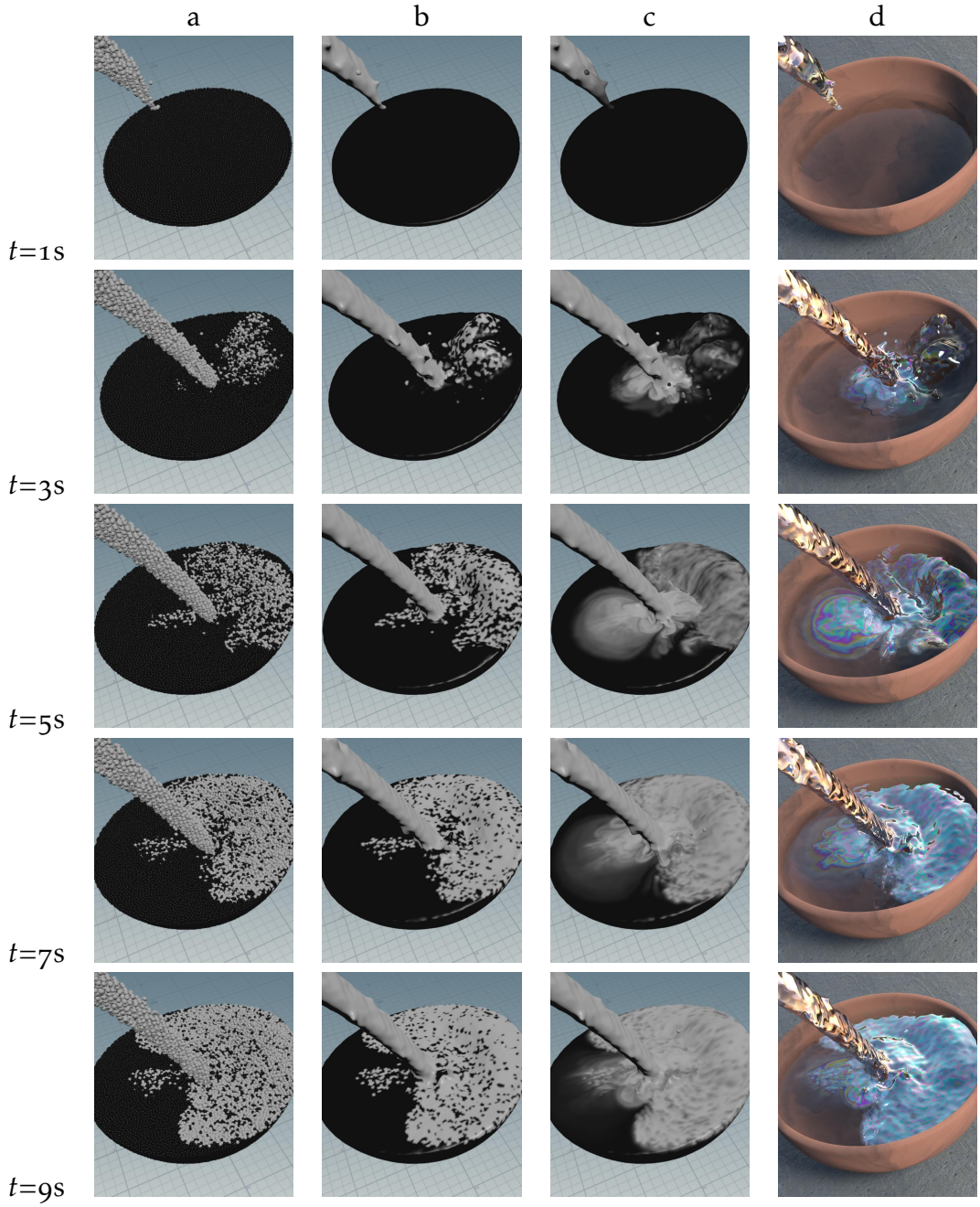
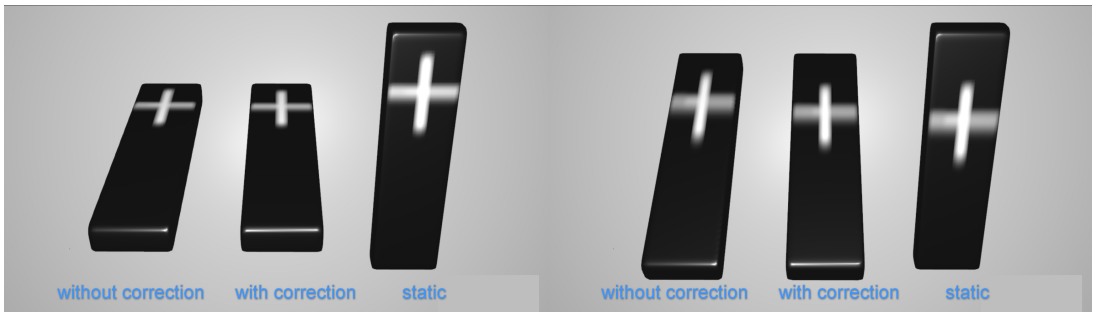
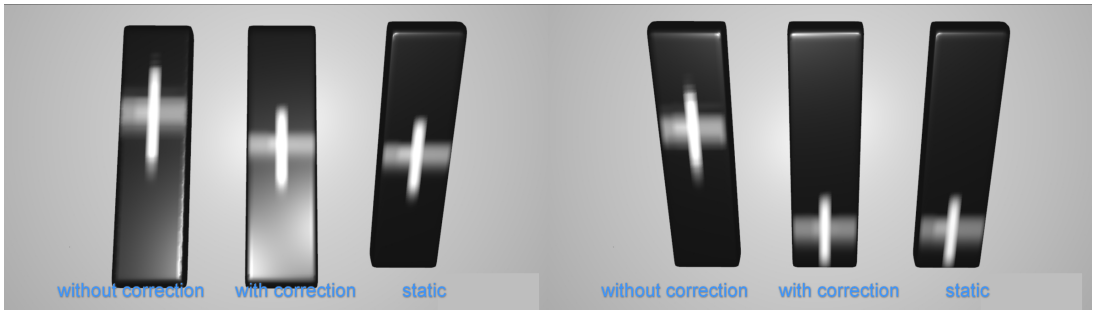


Table B.2: Image sequence for pouring polluted water into a bowl. The first column (a) shows the source SPH particles, column (b) the original scalar field derived from particle attributes. Column (c) shows the resulting 2D simulation on the surface and the last column (d) the resulting rendering. Time t is measured in seconds.

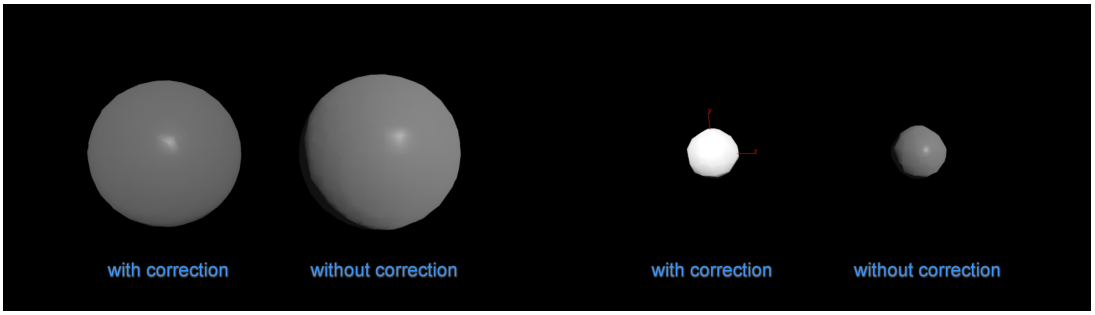


(a) t=0 (b) t=1s

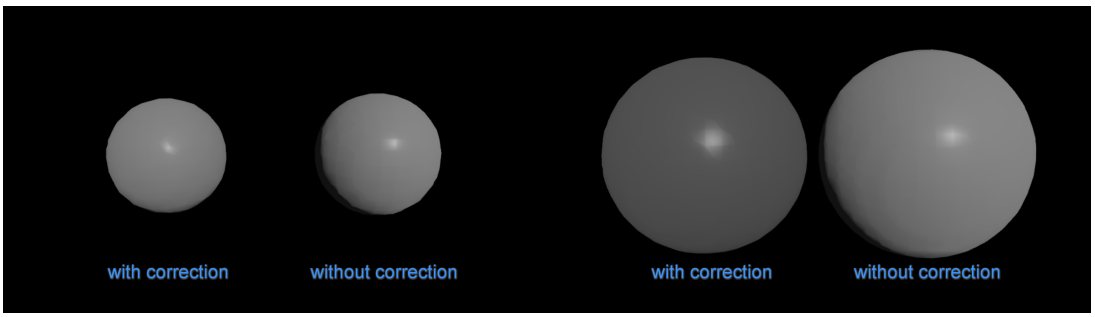


(c) t=2s (d) t=3s

Figure B.3: To demonstrate the effect of the vector length conservation, an experiment was performed with a rotating surface, where the surfactant was initialized with a tangential velocity, and then the surface was rotated 90 degrees. The surface was simulated with and without the length correction and a static surface as a reference side-by-side. Without length correction, the surface simulation shows much lower speeds of the imposed simulation and even stops in the middle of the surface; in the corrected version, the velocity is preserved during the rotation and matches the static surface.

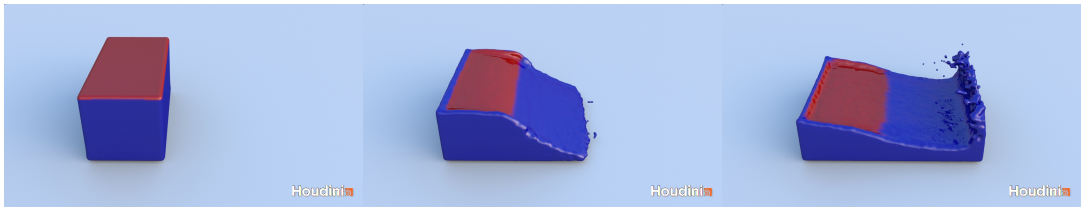


(a) t=0 (b) t=1s

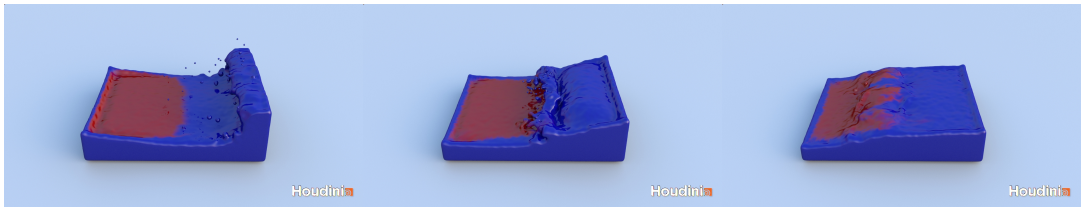


(c) t=2s (d) t=3s

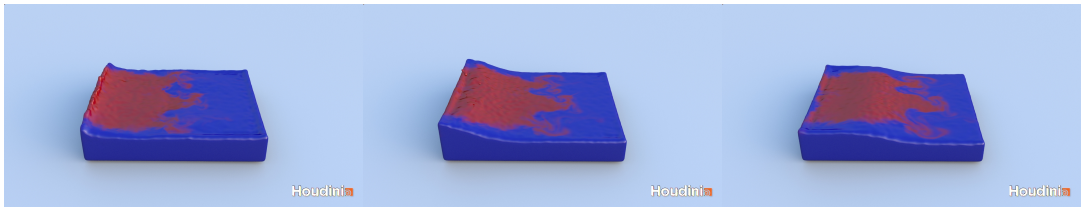
Figure B.4: A hand-animated sphere with a periodically oscillating radius was used to test the approach to mass conservation. On the left is the sphere with mass conservation. When the sphere is shrinking, the density on the sphere is increasing. When the sphere is growing, the density is decreasing. The sphere without correction has a constant density that does not react to changes of the surface area.



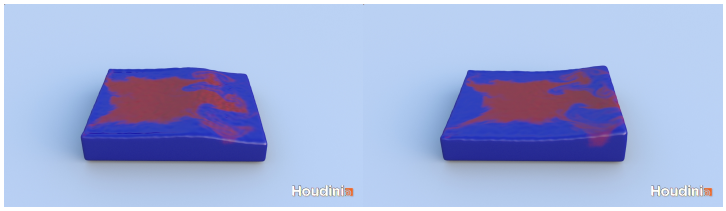
(a) t=0 (b) t=1s (c) t=2s



(d) t=3s (e) t=4s (f) t=5s

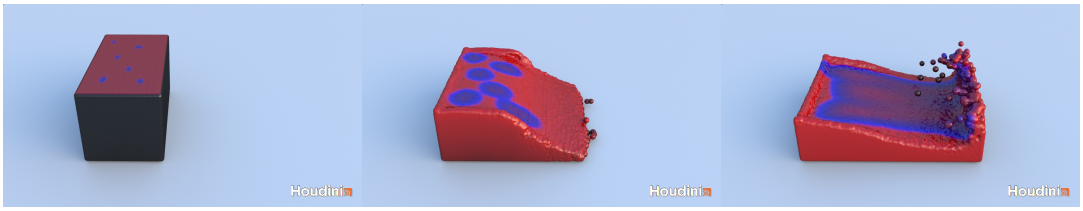


(g) t=6s (h) t=7s (i) t=8s

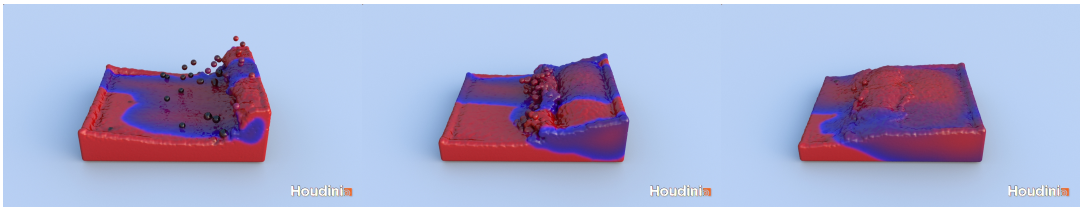


(j) t=9s (k) t=10s

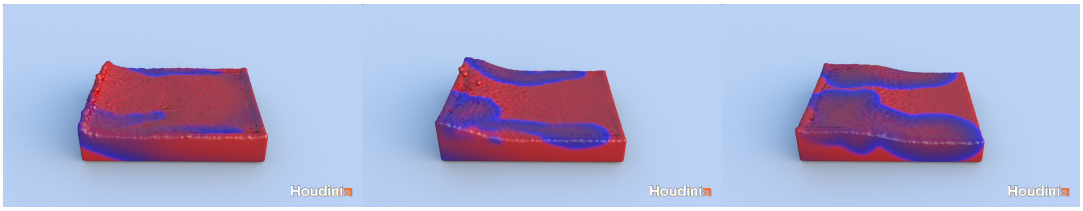
Figure B.5: Dam break simulation with added fluid simulation on the surface. Frames in 1 second steps.



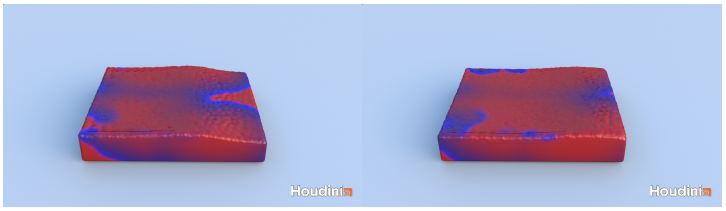
(a) t=0 (b) t=1s (c) t=2s



(d) t=3s (e) t=4s (f) t=5s

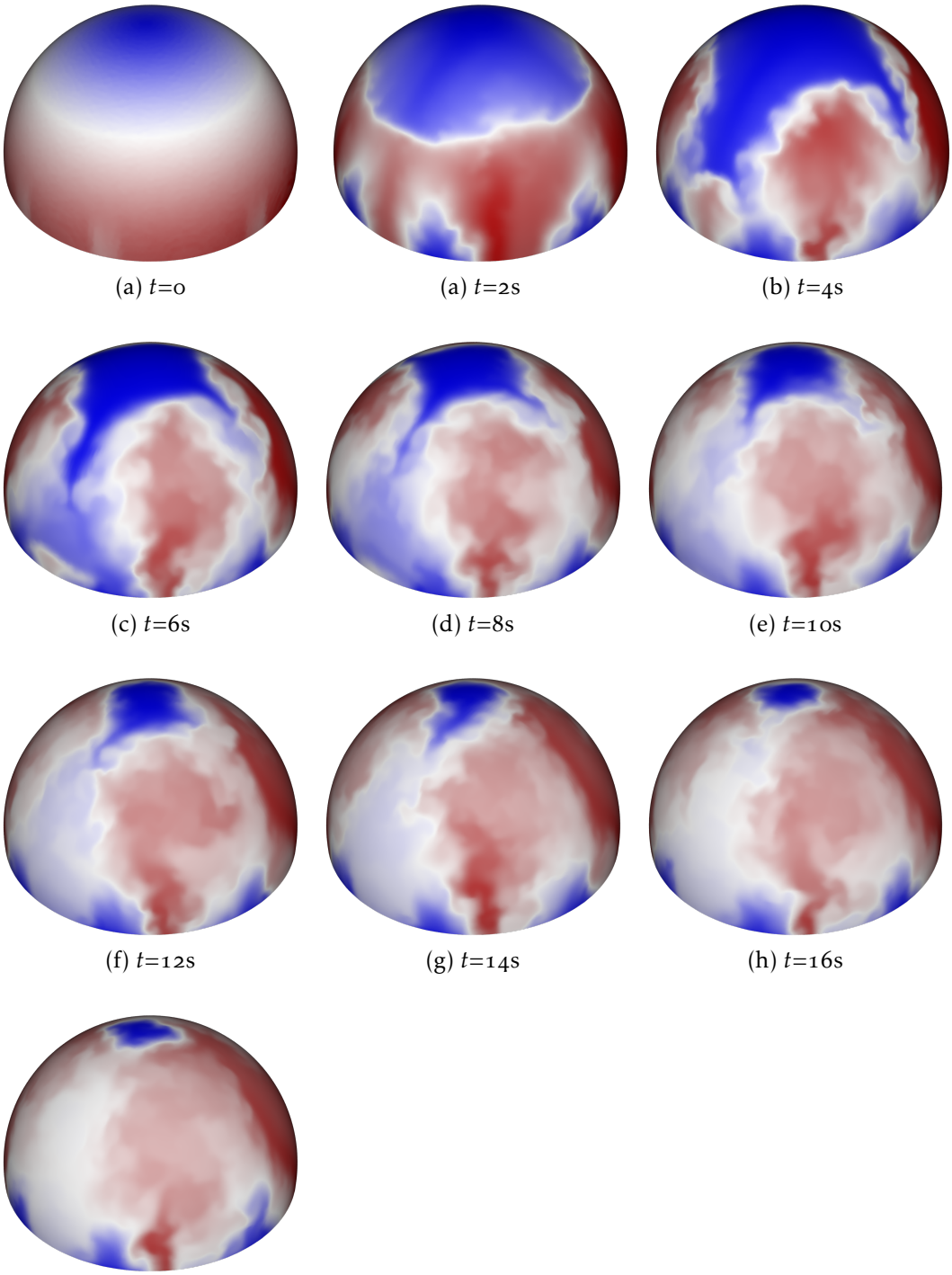


(g) t=6s (h) t=7s (i) t=8s



(j) t=9s (k) t=10s

Figure B.6: Dam break simulation with added reaction-diffusion simulation on the surface. Frames in 1 second steps.



(a) t=0 (a) t=2s (b) t=4s

(c) t=6s (d) t=8s (e) t=10s

(f) t=12s (g) t=14s (h) t=16s

(i) t=18s

Figure B.7: Sequence of thermal convection on a hemisphere as shown in Fig. 5.6. The temperature is color-coded, where the temperature rises from blue over white to red.

Co-Authored References

[1] D. Morgenroth, S. Reinhardt, D. Weiskopf, and B. Eberhardt, "Efficient 2D simulation on moving 3D surfaces," *Computer Graphics Forum*, vol. 39, no. 8, pp. 27–38, 2020. DOI: 10.1111/cgf.14098.

[2] D. Morgenroth, D. Weiskopf, and B. Eberhardt, "Distributed VFX architecture for SPH simulation," *Journal of WSCG*, vol. 21, no. 3, pp. 163–171, 2013. eprint: http://hdl.handle.net/11025/6986.

[3] D. Morgenroth, D. Weiskopf, and B. Eberhardt, "Direct raytracing of a closed-form fluid meniscus," *The Visual Computer*, vol. 32, no. 6-8, pp. 791–800, 2016. DOI: 10.1007/s00371-016-1258-4.

References

[4] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas, "Adaptively sampled particle fluids," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 48, 2007. DOI: 10.1145/1275808.1276437.

[5] A. W. Adamson, A. P. Gast, *et al.*, *Physical Chemistry of Surfaces*. Interscience Publishers New York, 1967, vol. 150. DOI: 10.1149/1.2133374.

[6] R. K. Agarwal, K.-Y. Yun, and R. Balakrishnan, "Beyond Navier–Stokes: Burnett equations for flows in the continuum–transition regime," *Physics of Fluids*, vol. 13, no. 10, pp. 3061–3085, 2001. DOI: 10.1063/1.1397256.

[7] G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner, "Parallel surface reconstruction for particle-based fluids," *Computer Graphics Forum*, vol. 31, no. 6, pp. 1797–1809, 2012. DOI: 10.1111/j.1467-8659.2012.02096.x.

[8] N. Akinci, A. Dippel, G. Akinci, and M. Teschner, "Screen space foam rendering," *Journal of WSCG*, vol. 21, no. 3, pp. 173–182, 2013. eprint: https://otik.uk.zcu.cz/handle/11025/6982.

[9] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser, "A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 583–592. DOI: 10.1109/PDP.2010.51.

[10] S. Auer, C. B. Macdonald, M. Treib, J. Schneider, and R. Westermann, "Real-time fluid effects on surfaces using the closest point method," *Computer Graphics Forum*, vol. 31, no. 6, pp. 1909–1923, 2012. DOI: 10.1111/j.1467-8659.2012.03071.x.

[11] S. Auer and R. Westermann, "A semi-Lagrangian closest point method for deforming surfaces," *Computer Graphics Forum*, vol. 32, no. 7, pp. 207–214, 2013. DOI: 10.1111/cgf.12228.

[12] Autodesk, *Autodesk 3ds Max Userguide*, http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/index.html Accessed: 2022-01-11.

[13] J. A. Bærentzen and H. Aanæs, "Generating signed distance fields from triangle meshes," Informatics and Mathematical Modelling, Technical University of Denmark, Tech. Rep. IMM-TR-2002-21, 2002.

[14] G. K. Batchelor, *An Introduction to Fluid Dynamics*. Cambridge University Press, 2000. DOI: 10.1017/CB09780511800955.

[15] M. Becker and M. Teschner, "Weakly compressible SPH for free surface flows.," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 24, 2007, pp. 209–217. eprint: https://dl.acm.org/doi/10.5555/1272690.1272719.

[16] J. Bender and D. Koschier, "Divergence-free smoothed particle hydrodynamics," in *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2015, pp. 147–155. DOI: 10.1145/2786784.2786796.

[17] M. Born and E. Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*. Elsevier, 2013. DOI: 10.1017/CB09781139644181.

[18] E. Bourque, J.-F. Dufort, M. Laprade, P. Poulin, and N. Chiba, "Simulating caustics due to liquid-solid interface menisci.," in *Proceedings of the Second Eurographics Conference on Natural Phenomena*, 2006, pp. 33–40. eprint: https://dl.acm.org/doi/abs/10.5555/2381370.2381376.

[19] J. Boussinesq, *Théorie de l'écoulement tourbillonnant et tumultueux des liquides dans les lits rectilignes a grande section*, 1st ed. Gauthier-Villars, 1897.

[20] J. U. Brackbill and H. M. Ruppel, "Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions," *Journal of Computational Physics*, vol. 65, no. 2, pp. 314–343, 1986. DOI: 10.1016/0021-9991(86)90211-1.

[21] J. Brooke, "SUS: a quick and dirty usability scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, Eds., Taylor and Francis, 1996.

[22] Chaos Software, *Phoenix product website*, https://www.chaosgroup.com/phoenix-fd/3ds-max Accessed: 2022-01-11.

[23] P. Clausen, M. Wicke, J. R. Shewchuk, and J. F. O'Brien, "Simulating liquids and solid-liquid interactions with Lagrangian meshes," *ACM Transactions on Graphics*, vol. 32, no. 2, 17:1–17:15, 2013. DOI: 10.1145/2451236.2451243.

[24] P. W. Cleary, S. H. Pyo, M. Prakash, and B. K. Koo, "Bubbling and frothing liquids," *ACM Transactions on Graphics*, vol. 26, no. 3, 97:2–97:6, 2007. DOI: 10.1145/1276377.1276499.

[25] M. Deserno, *The shape of a straight fluid meniscus*, http://www.cmu.edu/biolphys/deserno/pdf/meniscus.pdf Accessed: 2022-01-11, 2004.

[26] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 15–22. DOI: 10.1145/383259.383260.

[27] J.-C. Fernandez-Toledano, T. Blake, P. Lambert, and J. De Coninck, "On the cohesion of fluids and their adhesion to solids: Young's equation at the atomic scale," *Advances in Colloid and Interface Science*, vol. 245, pp. 102–107, 2017. DOI: 10.1016/j.cis.2017.03.006.

[28] P. Fournier, A. Habibi, and P. Poulin, "Simulating the flow of liquid droplets," in *Proceedings of the Graphics Interface 1998 Conference*, 1998, pp. 133–142.

[29] R. Fraedrich, S. Auer, and R. Westermann, "Efficient high-quality volume rendering of SPH data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1533–1540, 2010. DOI: 10.1109/TVCG.2010.148.

[30] F. Franks, *Water - A Comprehensive Treatise*. Springer Verlag US, 1975. DOI: 10.1007/978-1-4684-2958-9.

[31] J. Gagnon, F. Dagenais, and E. Paquette, "Dynamic lapped texture for fluid simulations," *The Visual Computer*, vol. 32, no. 6, pp. 901–909, 2016. DOI: 10.1007/s00371-016-1262-8.

[32] K. Garg and S. K. Nayar, "Photorealistic rendering of rain streaks," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 996–1002, 2006. DOI: 10.1145/1141911.1141985.

[33] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly Notices of the Royal Astronomical Society*, vol. 181, no. 3, pp. 375–389, 1977. DOI: 10.1093/mnras/181.3.375.

[34] P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola, "Interactive SPH simulation and rendering on the GPU," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010, pp. 55–64. eprint: https://dl.acm.org/doi/10.5555/1921427.1921437.

[35] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin, "Fitted BVH for fast raytracing of metaballs," *Computer Graphics Forum*, vol. 29, no. 2, pp. 281–288, 2010. DOI: 10.1111/j.1467-8659.2009.01597.x.

[36] O. Gourmel, L. Barthe, M.-P. Cani, B. Wyvill, A. Bernhardt, M. Paulin, and H. Grasberger, "A gradient-based implicit blend," *ACM Transactions on Graphics*, vol. 32, no. 2, 12:1–12:13, 2013. DOI: 10.1145/2451236.2451238.

[37] P. Gray and S. Scott, "Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Oscillations and instabilities in the system $A + 2B \rightarrow 3B$; B $\rightarrow$ C," *Chemical Engineering Science*, vol. 39, no. 6, pp. 1087–1097, 1984. DOI: 10.1016/0009-2509(84)87017-7.

[38] J. L. Gustafson, "Moore's law," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 1177–1184. DOI: 10.1007/978-0-387-09766-4_81.

[39] J. C. Hart, "Ray tracing implicit surfaces," *ACM SIGGRAPH 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pp. 1–16, 1993.

[40] W. Huang, J. Iseringhausen, T. Kneiphof, Z. Qu, C. Jiang, and M. B. Hullin, "Chemomechanical simulation of soap film flow on spherical bubbles," *ACM Transactions on Graphics*, vol. 39, no. 4, 2020. DOI: 10.1145/3386569.3392094.

[41] M. Huber, B. Eberhardt, and D. Weiskopf, "Boundary handling at cloth–fluid contact," *Computer Graphics Forum*, vol. 34, no. 1, pp. 14–25, 2015. DOI: 10.1111/cgf.12455.

[42] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner, "A parallel SPH implementation on multi-core CPUs," *Computer Graphics Forum*, vol. 30, no. 1, pp. 99–112, 2011. DOI: 10.1111/j.1467-8659.2010.01832.x.

[43] M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner, "Unified spray, foam and air bubbles for particle-based fluids," *The Visual Computer*, vol. 28, no. 6, pp. 669–677, 2012. DOI: 10.1007/s00371-012-0697-9.

[44] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner, "A parallel SPH implementation on multi-core CPUs," *Computer Graphics Forum*, vol. 30, no. 1, pp. 99–112, 2011. DOI: 10.1111/j.1467-8659.2010.01832.x.

[45] M. Ihmsen, J. Bader, G. Akinci, and M. Teschner, "Animation of air bubbles with SPH," in *Proceedings of the International Conference on Computer Graphics Theory and Applications*, 2011, pp. 225–234. DOI: 10.5220/0003322902250234.

[46] S. Ishida, P. Synak, F. Narita, T. Hachisuka, and C. Wojtan, "A model for soap film dynamics with evolving thickness," *ACM Transactions on Graphics*, vol. 39, no. 4, 31:1–31:11, 2020. DOI: 10.1145/3386569.3392405.

[47] Y. Jung and J. Behr, "GPU-based real-time on-surface droplet flow in X3D," in *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, 2009, pp. 51–54. DOI: 10.1145/1559764.1559772.

[48] K. Kaneda, S. Ikeda, and H. Yamashita, "Animation of water droplets moving down a surface," *The Journal of Visualization and Computer Animation*, vol. 10, no. 1, pp. 15–26, 1999. DOI: 10.1002/(SICI)1099-1778(199901/03)10:1%3C15::AID-VIS192%3E3.0.CO;2-P.

[49] K. Kaneda, T. Kagawa, and H. Yamashita, "Animation of water droplets on a glass plate," in *Proceedings of Computer Animation*, 1993, pp. 177–189. DOI: 10.1007/978-4-431-66911-1_17.

[50] P.-R. Kim, H.-Y. Lee, J.-H. Kim, and C.-H. Kim, "Controlling shapes of air bubbles in a multi-phase fluid simulation," *The Visual Computer*, vol. 28, no. 6, pp. 597–602, 2012. DOI: 10.1007/s00371-012-0696-x.

[51] T. Kim, J. Tessendorf, and N. Thuerey, "Closest point turbulence for liquid surfaces," *ACM Transactions on Graphics*, vol. 32, no. 2, 15:1–15:13, 2013. DOI: 10.1145/2451236.2451241.

[52] A. Kloeckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA: GPU run-time code generation for high-performance computing," *CoRR*, vol. abs/0911.3456, 2009. eprint: http://arxiv.org/abs/0911.3456.

[53] D. Koschier, C. Deul, M. Brand, and J. Bender, "An hp-adaptive discretization algorithm for signed distance field generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 10, pp. 2208–2221, 2017. DOI: 10.1109/TVCG.2017.2730202.

[54] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, "Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids," in *Eurographics 2019 - Tutorials*, 2019. DOI: 10.2312/egt.20191035.

[55] M. Lagergren, "GPU accelerated SPH simulation of fluids for VFX," Dept. Sci. Tech., Linköping University, Report LiU-ITN-TEK-A–10/044–SE, 2010.

[56] B. Lautrup, "Physics of continuous matter-exotic and everyday phenomena in the macroscopic world," *Applied Rheology*, vol. 15, no. 6, p. 369, 2005.

[57] C. H. Lewis, "Using the 'Thinking Aloud' method in cognitive interface design," IBM, Tech. Report RC-9265, 1982.

[58] Y. Liu, H. Zhu, X. Liu, and E. Wu, "Real-time simulation of physically based on-surface flow," *The Visual Computer*, vol. 21, no. 8-10, pp. 727–734, 2005. DOI: 10.1007/s00371-005-0314-2.

[59] J. A. Lock, C. L. Adler, D. Ekelman, J. Mulholland, and B. Keating, "Analysis of the shadow-sausage effect caustic," *Applied Optics*, vol. 42, no. 3, pp. 418–428, 2003. DOI: 10.1364/AO.42.000418.

[60] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987. DOI: 10.1145/280811.281026.

[61] O. Mercier, C. Beauchemin, N. Thuerey, T. Kim, and D. Nowrouzezahrai, "Surface turbulence for particle-based liquid simulations," *ACM Transactions on Graphics*, vol. 34, no. 6, 202:1–202:10, 2015. DOI: 10.1145/2816795.2818115.

[62] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, pp. 543–574, 1992. DOI: 10.1146/annurev.aa.30.090192.002551.

[63] D. Morgenroth, *Video appendix for enhancing fluid animation with fine detail*, 2020. DOI: 10.5281/zenodo.5517314.

[64] D. Morgenroth, S. Reinhardt, D. Weiskopf, and B. Eberhardt, *Application and sample scenes for efficient 2D simulation on moving 3D surfaces*, 2020. DOI: 10.5281/zenodo.4009208.

[65] D. Morgenroth, S. Reinhardt, D. Weiskopf, and B. Eberhardt, *Source code for efficient 2D simulation on moving 3D surfaces*, https://github.com/dimo3d/Cappucino Accessed: 2022-01-11, 2020.

[66] G. Morton, "A computer oriented geodetic database; and a new technique in file sequencing," *IBM Ltd, Ottawa, Technical Report.*, 1966.

[67] M. Mueller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003, pp. 154–159. eprint: http://dl.acm.org/citation.cfm?id=846276.846298.

[68] K. Museth, "VDB: high-resolution sparse volumes with dynamic topology," *ACM Transactions on Graphics*, vol. 32, no. 3, 27:1–27:22, 2013. DOI: 10.1145/2487228.2487235.

[69] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce, "OpenVDB: An open-source data structure and toolkit for high-resolution volumes," in *ACM SIGGRAPH 2013 Courses*, 2013, 19:1–19:1. DOI: 10.1145/2504435.2504454.

[70] N. Nakata, M. Kakimoto, and T. Nishita, "Animation of water droplets on a hydrophobic windshield," in *WSCG Conference Proceedings*, 2012, pp. 95–103.

[71] Next Limit, *Realflow product website*, http://www.realflow.com Accessed: 2022-01-11.

[72] J. Nielsen, "Estimating the number of subjects needed for a thinking aloud test," *International Journal of Human-Computer Studies*, vol. 41, no. 3, pp. 385–397, 1994. DOI: 10.1006/ijhc.1994.1065.

[73] A. Nilsson, *Long lost pen pal*, https://www.flickr.com/photos/andreasnilsson1976/530776998/ Accessed: 2022-01-11, License: https://creativecommons.org/licenses/by-nc-nd/2.0/.

[74] E. G. Parker and J. F. O'Brien, "Real-time deformation and fracture in a game environment," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 156–166. DOI: 10.1145/1599470.1599492.

[75] O. Redlich, "Intensive and extensive properties," *Journal of Chemical Education*, vol. 47, no. 2, pp. 154–156, 1970. DOI: 10.1021/ed047p154.2.

[76] S. Reinhardt, T. Krake, B. Eberhardt, and D. Weiskopf, "Consistent Shepard interpolation for SPH-based fluid animation," *ACM Transaction on Graphics*, vol. 38, no. 6, 189:1–189:11, 2019. DOI: 10.1145/3355089.3356503.

[77] S. J. Ruuth and B. Merriman, "A simple embedding method for solving partial differential equations on surfaces," *Journal of Computational Physics*, vol. 227, no. 3, pp. 1943–1961, 2008. DOI: 10.1016/j.jcp.2007.10.009.

[78] F. S., *Msgpack website*, http://msgpack.org Accessed: 2022-01-11.

[79] G. Saville, "Computer simulation of the liquid–solid–vapour contact angle," *Journal of the Chemical Society, Faraday Transactions 2*, vol. 73, pp. 1122–1132, 1977. DOI: 10.1039/F29777301122.

[80] F. Seychelles, Y. Amarouchene, M. Bessafi, and H. Kellay, "Thermal convection and emergence of isolated vortices in soap bubbles," *Physical Review Letters*, vol. 100, no. 14, p. 144 501, 2008. DOI: 10.1103/PhysRevLett.100.144501.

[81] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, 1994. eprint: https://dl.acm.org/doi/10.5555/865018.

[82] L. Shi and Y. Yu, "Inviscid and incompressible fluid simulation on triangle meshes," *Computer Animation and Virtual Worlds*, vol. 15, no. 3-4, pp. 173–181, 2004. DOI: 10.1002/cav.19.

[83] E. Shirani and S. Jafari, "Application of lbm in simulation of flow in simple micro-geometries and micro porous media," *African Physical Review*, vol. 1, 2007.

[84] J. M. Singh and P. Narayanan, "Real-time ray tracing of implicit surfaces on the GPU," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 261–272, 2010. DOI: 10.1109/TVCG.2009.41.

[85] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible SPH," *ACM Transactions on Graphics*, vol. 28, no. 3, 40:1–40:6, 2009. DOI: 10.1145/1531326.1531346.

[86] B. Solenthaler, J. Schläfli, and R. Pajarola, "A unified particle model for fluid–solid interactions," *Computer Animation and Virtual Worlds*, vol. 18, no. 1, pp. 69–82, 2007. DOI: 10.1002/cav.162.

[87] M. Spivak, *A Comprehensive Introduction to Differential Geometry*. Publish or Perish, 1999, vol. 4.

[88] J. Stam, "Stable fluids," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 1999, pp. 121–128. DOI: 10.1145/311535.311548.

[89] J. Stam, "Flows on surfaces of arbitrary topology," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 724–731, 2003. DOI: 10.1145/1201775.882338.

[90] I. Stuppacher and P. Supan, "Rendering of water drops in real-time," in *Central European Seminar on Computer Graphics for Students*, 2007.

[91] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha, "Interactive 3D distance field computation using linear factorization," in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, 2006, pp. 117–124. DOI: 10.1145/1111411.1111432.

[92] L. Szirmay-Kalos and T. Umenhoffer, "Displacement mapping on the GPU – state of the art," *Computer Graphics Forum*, vol. 27, no. 6, pp. 1567–1592, 2008. DOI: 10.1111/j.1467-8659.2007.01108.x.

[93] T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki, "Realistic animation of fluid with splash and foam," *Computer Graphics Forum*, vol. 22, no. 3, pp. 391–400, 2003. DOI: 10.1111/1467-8659.00686.

[94] S. Takenaka, Y. Mizukami, and K. Tadamura, "A fast rendering method for water droplets on glass surfaces," in *The 23rd International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2008, pp. 13–16.

[95] J. Tessendorf, "Simulating ocean water," *SIGGRAPH Course Notes*, 2001.

[96] N. Thuerey, F. Sadlo, S. Schirm, M. Müller-Fischer, and M. Gross, "Real-time simulations of bubbles and foam within a shallow water framework," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 191–198. DOI: 10.2312/SCA/SCA07/191-198.

[97] I. R. Türkmen, "Homogeneous nucleation rates of ice in supercooled binary liquid mixtures of water + non-electrolytes: A combined theoretical and experimental study," Ph.D. dissertation, Freie Universität Berlin, 2007. DOI: http://dx.doi.org/10.17169/refubium-6006.

[98] D. Valdez-Balderas, J. M. Domínguez, B. D. Rogers, and A. J. C. Crespo, "Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters," *Journal of Parallel and Distributed Computing*, 2012. DOI: 10.1016/j.jpdc.2012.07.010.

[99] H. Wang, P. J. Mucha, and G. Turk, "Water drops on surfaces," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 921–929, 2005. DOI: 10.1145/1186822.1073284.

[100] L. Wang, W. Chen, W. Yang, F. Bi, and F. R. Yu, "A state-of-the-art review on image synthesis with generative adversarial networks," *IEEE Access*, vol. 8, pp. 63 514–63 537, 2020. DOI: 10.1109/ACCESS.2020.2982224.

[101] F. Wu and C. Zheng, "Illumination model for two-layer thin film structures," in *GRAPP*, 2015, pp. 199–206. DOI: 10.5220/0005261401990206.

[102] H. Xu and J. Barbič, "Signed distance fields for polygon soup meshes," in *Proceedings of Graphics Interface*, 2014, pp. 35–41. DOI: 10.1201/9781003059325-5.

[103] J.-J. Xu and H.-K. Zhao, "An Eulerian formulation for solving partial differential equations along a moving interface," *Journal of Scientific Computing*, vol. 19, no. 1, pp. 573–594, 2003. DOI: 10.1023/A:1025336916176.

[104] H. Yan, Z. Wang, J. He, X. Chen, C. Wang, and Q. Peng, "Real-time fluid simulation with adaptive SPH," *Computer Animation and Virtual Worlds*, vol. 20, pp. 417–426, 2009. DOI: 10.1002/cav.v20:2/3.

[105] T. Young, "An essay on the cohesion of fluids," *Philosophical Transactions of the Royal Society of London*, vol. 95, pp. 65–87, 1805.

[106] J. Yu and G. Turk, "Reconstructing surfaces of particle-based fluids using anisotropic kernels," *ACM Transactions on Graphics*, vol. 32, no. 1, 5:1–5:12, 2013. DOI: 10.1145/2421636.2421641.

[107] Y.-J. Yu, H.-Y. Jung, and H.-G. Cho, "A new water droplet model using metaball in the gravitational field," *Computers & Graphics*, vol. 23, no. 2, pp. 213–222, 1999. DOI: 10.1016/S0097-8493(99)00031-X.

[108] Y. Yuan and T. R. Lee, "Contact angle and wetting properties," in *Surface Science Techniques*, G. Bracco and B. Holst, Eds., Heidelberg: Springer, 2013, pp. 3–34. DOI: 10.1007/978-3-642-34243-1_1.

[109] Y. Zhang, "Adaptive sampling and rendering of fluids on the GPU," in *Proceedings Symposium on Point-Based Graphics*, 2008, pp. 137–146. DOI: https://doi.org/10.5167/uzh-9735.

[110] Y. Zhang, H. Wang, S. Wang, Y. Tong, and K. Zhou, "A deformable surface model for real-time water drop animation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 8, pp. 1281–1289, 2012.

[111] H.-K. Zhao, B. Merriman, S. Osher, and L. Wang, "Capturing the behavior of bubbles and drops using the variational level set approach," *Journal of Computational Physics*, vol. 143, no. 2, pp. 495–518, 1998. DOI: 10.1006/jcph.1997.5810.

[112] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 965–972, 2005. DOI: 10.1145/1073204.1073298.

[113] Q. Zia Ullah, S. Hassan, and G. M. Khan, "Adaptive resource utilization prediction system for infrastructure as a service cloud," *Computational Intelligence and Neuroscience*, vol. 2017, 2017. DOI: 10.1155/2017/4873459.