Particle-based Simulation of Large Bodies of Water with Bubbles, Spray and Foam

vorgelegt von

Markus Ihmsen

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät,
Albert-Ludwigs-Universität Freiburg im Breisgau

September 2013

1. Gutachter	Prof. Dr. Matthias Teschner
2. Gutachter	Prof. Dr. Rüdiger Westermann
Dekan	Prof. Dr. Yiannos Manoli
Datum der Disputation	24.09.2013

Abstract

In the field of computational fluid dynamics, either particle-based or mesh-based models are employed to discretize the equations of motion. Particle-based methods have some advantages over mesh-based approaches, particularly in their ability to handle free surfaces and interfaces with other materials, as well as in tracking splashes and droplets. Accordingly, particle solvers are an attractive tool for simulating fluids. However, for simulations of large bodies of water, methods that operate exclusively on particles are currently not favored. In order to solve the governing equations, sets of interacting particles must be queried and processed multiple times per simulation step. Furthermore, enforcing incompressibility is essential to obtain a realistic simulation of water. However, this is considered to be numerically challenging in particle-based methods.

This thesis addresses these challenges of pure particle-based methods. Various contributions are made which in sum yield an efficient framework for simulations of water using the Smoothed Particle Hydrodynamics (SPH) method. First of all, this thesis proposes a novel incompressible SPH solver which is based on a novel discretized form of the pressure Poisson equation. The proposed formulation outperforms the pressure computation of previous SPH solvers, significantly. Large time steps and small density deviations of down to 0.01% can be handled in typical scenarios. Furthermore, the efficient querying and processing of particle neighbors is addressed. Therefore, two spatial acceleration structures are proposed which employ temporal coherence and preserve spatial locality. As the design of the data structures ensures a low memory traffic, the proposed structures scale well on parallel architectures. The performance, quality and robustness of the simulation is also influenced by the handling of interfaces. This thesis presents a pressure-based boundary scheme which obtains smooth pressure and force fields at the fluid-solid interface. Non-penetration of arbitrarily shaped boundaries is enforced even for comparatively large time steps. The practicability of the proposed techniques is demonstrated by a variety of large-scale scenarios with up to 40 million SPH particles.

The interplay of air and water is often neglected in the field of computer graphics as the high density ratio of water to air is numerically challenging. However, phenomena like air bubbles, foam, mist or spray are omnipresent in turbulent flow. Therefore, simulating the formation and transport of air-water mixtures can significantly enhance the realism of fluid simulations. This thesis proposes a particle-based technique to simulate the phenomena caused by the interplay of air and water. Thereby, the entrapment of air and prominent whitewater effects, e.g., in front of breaking waves can be animated.

In physically-based simulation techniques, the accuracy scales with the resolution. However,

higher resolutions induce longer computation times, particularly as the practical time step scales inversely with the resolution. In this thesis, a multi-resolution framework is presented where the numerically critical forces are computed on a coarse scale, while visual detail is added in a secondary simulation to an existing simulation by upsampling the simulated material.

Zusammenfassung

In der numerischen Strömungsmechanik werden entweder partikelbasierte oder gitterbasierte Modelle benutzt, um die Bewegungsgleichungen zu diskretisieren. Partikelbasierte Modelle haben einige Vorteile gegenüber gitterbasierten Ansätzen, besonders bei der Behandlung freier Oberflächen und an Grenzflächen mit anderen Materialien, sowie im Verfolgen von Spritzwasser und einzelner Tropfen. Daher bilden Partikel-Methoden eine attraktive Möglichkeit zur Simulation von Fluiden. Für Simulationen großer Wassermengen werden Methoden, die ausschließlich mit Partikeln arbeiten, momentan nicht bevorzugt. Um die zugrundeliegenden Gleichungen zu lösen, müssen in jedem Simulationsschritt interagierende Partikelmengen gefunden und mehrfach verarbeitet werden. Außerdem ist das Einhalten des Volumens eine essentielle Voraussetzung, um realistische Wassersimulationen zu erzeugen. Das Einhalten dieser Bedingung gilt für Partikelmethoden als eine große numerische Herausforderung.

Diese Herausforderungen rein partikelbasierter Methoden sind Inhalt dieser Dissertation. Eine Vielzahl von Beiträgen wird geleistet, die in Summe ein effizientes System für die Simulation von Wasser ermöglichen, unter der Verwendung der Smoothed Particle Hydrodynamics (SPH) Methode. Zunächst wird eine neue SPH-Methode zur Berechnung von inkompressiblen Flüssigkeiten vorgeschlagen. Diese Methode basiert auf einer neuen Diskretisierung der Poisson-Gleichung für den Druck. Die vorgestellte Formulierung übertrifft die Effizienz bisheriger SPH-Methoden deutlich. In typischen Szenarien kann mit verhältnismäßig großen Zeitschritten und geringen Dichteabweichungen von bis zu 0.01% gerechnet werden. Die Dissertation behandelt außerdem das effiziente Suchen und Bearbeiten von Partikelpaaren. Hierzu werden zwei Datenstrukturen vorgeschlagen, die zeitliche Kohärenz ausnutzen und räumliche Lokalität erhalten, um die räumliche Suche zu beschleunigen. Da das Design der Datenstrukturen einen geringen Datenverkehr erzwingt, skalieren die vorgeschlagenen Strukturen sehr gut auf parallelen Architekturen. Die Berechnungszeit, Qualität und Stabilität der Simulation wird auch durch die Methode zur Behandlung der Grenzflächen beeinflusst. Diese Dissertation präsentiert eine druckbasierte Methode, die an den Grenzflächen mit Festkörpern eine relativ gleichmäßige Verteilung des Druck- und Kraftfeldes erzeugt. Dadurch kann das Nichteindringen von Flüssigkeit in beliebig geformte Festkörper sichergestellt werden, auch für vergleichsweiße große Zeitschritte. Die Anwendbarkeit der vorgeschlagenen Techniken wird an einer Vielzahl von komplexen Szenen mit bis zu 40 Millionen SPH Partikeln demonstriert.

Die Wechselwirkung von Luft und Wasser wird in der Computergrafik meist vernachlässigt, da der hohe Dichteunterschied zwischen den beiden Phasen numerische Schwierigkeiten

bereitet. Jedoch sind Phänomene wie Luftblasen, Meeresschaum, Dunst und Gischt in turbulenten Flüssigkeiten allgegenwärtig. Durch das Simulieren der Bildung und Bewegung von Luft-Wasser-Vermischungen könnten der Simulation daher typische Effekte hinzugefügt werden. Dies gilt für kleine und große Wassermengen. Diese Dissertation schlägt eine effiziente Technik für partikelbasierte Methoden vor, um die Wechselwirkungen von Luft und Wasser zu simulieren. Dadurch können Lufteinschlüsse und typische Weißwassereffekte simuliert werden, wie z.B. Schaum, der von einer Welle angespült wird.

Die Genauigkeit physikalisch-basierter Simulationen skaliert mit der Auflösung. Höhere Auflösungen implizieren eine längere Berechnungszeit, insbesondere da der anwendbare Zeitschritt kleiner wird. In dieser Dissertation wird ein mehrstufiges System vorgeschlagen, bei dem die numerisch kritischen Kräfte für eine grobe Auflösung berechnet werden. Der Detailgrad wird in einer zweiten, hochaufgelösten Simulation erhöht. Dies erfolgt durch Upsampling des niedrig aufgelösten Materials.

Acknowledgements

First and foremost, I would like to thank Prof. Dr. Matthias Teschner for the excellent supervision. I have been extremely lucky to have had him as a mentor who cared so much about my work, and always had an open ear for all kind of questions and queries which he always tried to answer promptly. His passionate professional attitude has always been encouraging and has contributed a lot in realizing this thesis. I would also like to thank Prof. Dr. Rüdiger Westermann from the TU Munich for his interest in reading and reviewing this thesis.

During my Ph.D. studies I had the joy to work with highly skilled and friendly people. My former colleagues Dr. Jonas Spillmann, Dr. Markus Becker, Dr. Marc Gissler and Dr. Rüdiger Schmedding have always been very helpful and patient, particularly in the early days of my research career. I am heavily indebted to those group members that I worked closely with, namely: Gizem Akinci, Nadir Akinci and Jens Cornelis. I have always been proud to be able to work with such enormously motivated and talented people.

I would like to express my gratitude to Cynthia Findlay and Veria Popa who handled all the administrative tasks, and to Stefan Teister for caring about all the technical problems.

I want to thank Ariane, my wife, for backing me up during all the ups and downs of my research. Finally, I would like to thank my faithful mother who encouraged me to study computer science and to pursue my Ph.D..

This thesis has been supported by the German Research Foundation (DFG) under contract number TE 6321-2.

Contents

CHAPTER 1

Introduction

In the field of computer animation, fluid simulation has attracted many researchers and artists in the past two decades. Applications are ranging from interactive training environments to advertisement and feature films. Opposed to mechanical engineering, where the main goal is to achieve physical accuracy, the focus is set on visual plausibility, efficiency and stability. In interactive applications such as medical simulators or games, a pre-defined update rate is required to guarantee a satisfying user experience. For commercials and films, other requirements must be met. For digital compositing, the fluid animation should often be indistinguishable from reality and, thus, high-resolution fluids are required to capture all prominent effects. On the other hand, the animator should be in high control in order to guide the animation to the desired result.

Various methods have been employed to model complex fluid behavior while suiting the respective requirements. For simulating convincing ocean surfaces or shallow water, it might be sufficient to represent the fluid as a height field [Tes04]. This coarse approximation animates plausible wave motions at interactive rates. However, it rules out many interesting effects such as vortices or splashes. Higher realism is achieved by discretizing the fluid volume and solving the governing equations numerically. Therefore, two discretization schemes exist: the Eulerian and the Lagrangian model. In Eulerian methods, the fluid quantities are measured and evolved at fixed points in the simulation space. The grid discretization eases the numerical approximation of spatial derivatives and, therefore, simplifies the formulation and solution of the governing equations. On the other hand, the grid representation presents some challenges. Foremost, advecting the grid quantities imposes crucial numerical problems. Commonly, these problems are avoided by doing the advection either with implicit particles [Sta99] or by using explicit particles, e.g., FLIP [ZB05]. In both cases, averaging operations are required for interpolating the quantities from particles to the grid and back. At least for the implicit method, the averaging causes numerical dissipation, i.e., sharp features are smoothed out. Another major challenge is the extrapolation of values at the free surface and at the interface with complex and non-grid aligned boundaries. In contrast, the Lagrangian approach treats the fluid volume as a system of particles which can move freely and transport the fluid quantities. Thereby, the fluid volume can be tracked naturally.

Although Lagrangian methods have some benefits over Eulerian methods, they are not favored in computer graphics due to efficiency. The discretization of the spatial derivatives is computationally more expensive on an unstructured set of particles as typically more samplings points are required. Furthermore, the sampling points are not fixed in space. Accordingly, sets of interacting particles must be queried in each time step. Finally, in order to enforce incompressibility, the neighbor sets are processed multiple times which requires an efficient way of computing or accessing proximity-related information such as distance.

Figure 1.1: Left: A highly resolved simulation of a low-viscous liquid poured into a glass. The volume radius of the underlying simulation particles was set to 1mm. Right: Simulation of breaking waves with whitewater in a terrain of 1200m^2 floor area. The total fluid volume is discretized with 16 million SPH particles and up to 25 million secondary particles representing foam, bubbles and spray. The volume radius of SPH particles was set to 1.8cm.

1.1 Contributions

This thesis addresses the challenges of pure Lagrangian methods with respect to computational efficiency using the Smoothed Particle Hydrodynamics (SPH) method. It presents an highly efficient animation loop which makes it feasible to simulate large bodies of water comprising millions of particles and small scale scenarios at high resolution. Therefore, it introduces parallel data structures to enhance the computational efficiency and presents techniques which allow for large time steps. The realism of animations is improved by new methods for simulating the interplay of air and water, modeling bubbles, spray and foam. The presented set of techniques are enabling an SPH solver to capture a wide range of fluid phenomena ranging from air bubbles streaming around inflows to breaking waves with whitewater. First examples are given in Figure 1.1. Generally, realism can be traded for speed by changing the discretization scale. A finer discretization of the simulation space increases the quality of the animation while it decreases performance. In this thesis, a multi-scale approach is presented where the numerically critical forces are computed on a coarse scale at high update rates. Visual detail is added by coupling a secondary, highly resolved simulation to the coarse simulation. The method is applied for animations of granular material.

Incompressibility

Enforcing incompressibility is indispensable to achieve visually plausible animations of water. However, guaranteeing small to zero fluctuations of the fluid volume is challenging with SPH. There exist different strategies to address this issue, e.g., projection schemes [CR99, SL03], state-equation-based approaches (SESPH) [MCG03, BT07], predictor-corrector methods (PCISPH) [SP09] and position-based fluids (PBF) [MM13]. Among these, PCISPH and PBF are the most efficient methods since they can handle relatively large time steps while being comparably efficient to compute. In contrast, projection schemes are currently considered less practical as solving the pressure Poisson equation (PPE) demands for a numerically challenging approximation of the Laplacian in SPH. This thesis proposes a novel formula-

tion of the projection method for SPH. By combining a symmetric SPH pressure force and an SPH discretization of the continuity equation, a novel discretized form of the PPE is obtained. The novel formulation outperforms the pressure computation of previous projection schemes and PCISPH. Large time steps and small density deviations of down to 0.01% can be handled in typical scenarios. The practicability of the proposed algorithm is demonstrated by showing scenarios with up to 40 million SPH particles.

Neighborhood Search

In SPH, particles do not have any fixed connectivity. This requires the list of interacting particles to be updated in each simulation step. Therefore, the efficient querying and processing of particle neighbors is crucial for the performance of the simulation. In this thesis, two spatial acceleration structures are proposed which employ temporal coherence and preserve spatial locality. In the context of SPH, the performance of the presented data structures is thoroughly analyzed and compared with three existing variants of uniform grids, i.e., basic uniform grid, spatial hashing and index sort. The design of the data structures ensures a high cache-hit rate and, thus, a low memory traffic. Therefore, the proposed structures scale well on parallel architectures. This is demonstrated by applications on multi-core CPUs for the animation loop, as well as on many-core GPUs for surface reconstruction.

Boundary Handling

SPH employs the concept of integral representations of field quantities. The integrals are locally approximated as the Riemann sum over the finite set of sampling points, i.e., particles located within the support domain. Inside the fluid volume, the integral domain is well defined which means that the sampling is sufficient to obtain a good approximation of the quantity. However, special considerations have to be taken into account at the free surface and at the interface with solid objects in order to prevent spatial and temporal discontinuities of physical properties. Discontinuities might lead to perceivable simulation artifacts and numerical instabilities, particularly at the fluid-solid interface. Accordingly, the boundary-handling scheme influences the quality and robustness of the simulation. This thesis presents a pressure-based boundary scheme which obtains smooth quantity fields at the fluid-solid interface. Non-penetration of arbitrary shaped boundaries is enforced even for comparatively large time steps.

Multi-phase Fluids

Whenever a liquid is poured into a glass, air is likely to be trapped inside the fluid, creating bubbles. Thus, the visual realism of fluid animations can be significantly enhanced by modeling the entrapment of air in water. This thesis presents a novel SPH model for two-way coupled air-water simulations. High density ratios are handled by treating the two phases separately. Realistic drag effects are achieved by coupling water and air via the velocity field. Trapped air is simulated without explicitly modeling the air surrounding the fluid by generating air bubbles on the fly in regions with high velocity differences. The model overcomes stability issues, can handle large time steps and is efficient to compute.

Secondary Simulation

The interplay of air and water does not only create bubbles, but also causes diffuse material perceived as foam and spray. Simulating the formation and transport of diffuse material adds visual detail to large-scale fluid simulations such as breaking waves. However, for convincing results, a high resolution is required to capture the small-scale details, while for the bulk of the fluid a coarser resolution might be sufficient to capture the flow. This thesis presents a novel technique for adding diffuse material to particle-based simulations. This is done in a unified way instead of employing a set of techniques for different types of air-water mixtures. Prominent whitewater effects, e.g., in front of breaking waves can be animated. The model is realized as a post-processing step on pre-computed fluid simulations. It is shown that visually plausible results can be realized even though the influence of air on water is neglected.

Multiple Resolutions

The realism of the simulation increases with the resolution. On the other hand, higher resolutions also induce longer computation times. This is not only due to an increase in the number of particles, but also caused by the fact that each particle represents a smaller volume which is a limiting factor for the time step according to the CFL condition. Nevertheless, for convincing simulations of materials like sand a relatively high resolution is required. As modeling the physical behavior by simulating each physical grain imposes prohibitively high computational costs, an alternative method is required. This thesis presents a multi-resolution framework, where the mechanical behavior of the material is simulated on a coarse scale using a continuum approach which is based on SPH. Dispersion effects and proper interactions with complex geometries are simulated on a high-resolution simulation which is coupled to the base simulation in a post-process. Thereby, visual detail is added which could not be captured by the base simulation. The presented pipeline is very efficient as different spatial and temporal resolutions are employed. Numerically critical forces are computed only for a comparatively small set of coarsely resolved particles, while the highly resolved set of particles can be updated with a large time step of 10ms.

1.2 Publications

This thesis is based on the following peer-reviewed publications in journals:

[**IABT11**] M. Ihmsen, N. Akinci, M. Becker, M. Teschner, **"A Parallel SPH Implementation on Multi-core CPUs"**. *Computer Graphics Forum*, vol. 30, no. 1, pp. 99-112, 2011.

[**IAAT12**] M. Ihmsen, N. Akinci, G. Akinci, M. Teschner, **"Unified Spray, Foam and Bubbles for Particle-based Fluids"**. *The Visual Computer (Proc. CGI 2012)*, vol.28, no.6-8, pp. 669-677, 2012.

[**IWT13**] M. Ihmsen, A. Wahl, M. Teschner, **"A Lagrangian Framework for Simulating Granular Material with High Detail"**. *Computers & Graphics*, vol. 38, no. 5, pp.1-10, 2013.

[**ICS+13**] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, M. Teschner, **"Implicit Incompressible SPH"**. IEEE Transactions on Visualization and Computer Graphics. 2013, doi:10.1109/TVCG.2013.105.

[**AIAT12**] G. Akinci, M. Ihmsen, N. Akinci, M. Teschner, **"Parallel Surface Reconstruction for Particle-based Fluids"**. *Computer Graphics Forum*, vol. 31, no.6, pp. 1797-1809, 2012.

[**AIS+12**] N. Akinci, M. Ihmsen, B. Solenthaler, G. Akinci, M. Teschner, **"Versatile Rigid-Fluid Coupling for Incompressible SPH"**. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 62:1-62:8, 2012.

and peer-reviewed conference proceedings:

[**IAGT10**] M. Ihmsen, N. Akinci, M. Gissler, M. Teschner, **"Boundary Handling and Adaptive Time-stepping for PCISPH"**. *Proc. VRIPHYS*, pages 79–88, 2010.

[**IBAT11**] M. Ihmsen, J. Bader, G. Akinci, M. Teschner, **"Animation of Air Bubbles with SPH"**. *Proc. International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 225–234, 2011.

[**IWT12**] M. Ihmsen, A. Wahl, M. Teschner, **"High-Resolution Simulation of Granular Material with SPH"**. *Proc. VRIPHYS*, pp. 53-60, 2012. Best Paper Award.

[**BIT09**] M. Becker, M. Ihmsen, M. Teschner, **"Corotated SPH for deformable solids"**. *Proc. Eurographics Workshop on Natural Phenomena*, pages 27–34, 2009.

[**GIT11**] M. Gissler, M. Ihmsen, M. Teschner, **"Efficient Uniform Grids for Collision Handling in Medical Simulators"**. *Proc. International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 79–84, 2011.

[**AAIT12**] G. Akinci, N. Akinci, M. Ihmsen, M. Teschner, **"An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids"**. *Proc. VRIPHYS*, pp. 61-68, 2012.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 describes the theoretical foundation of SPH-based fluid solvers. The theory builds also the base for the contributions presented in the subsequent chapters. Chapter 3 presents a new incompressible SPH solver which is practical for simulating large-scale scenarios. In Chapter 4, efficient data structures and techniques are proposed for accelerating the neighborhood search on multi- and many-core architectures. The utility is demonstrated by applying the concept to the SPH simulation pipeline. The handling of interfaces with solid boundaries and other fluids imposes certain challenges which are addressed in Chapter 5 and Chapter 6, respectively.

Chapter 7 and Chapter 8 focus on efficient techniques for increasing the level of detail of pre-computed simulations. Chapter 7 proposes an effective model for simulating air-water mixtures perceived as foam, spray and bubble. The air-water material is represented by secondary particles which are added and simulated in a post-process. Due to its efficiency, generality, and the intuitive parameter setting, the model turns out to be an attractive tool for improving the visual realism of existing large-scale particle simulations. In some cases, detail might be captured not by just adding secondary material to an existing simulation, but by upsampling the simulated material in a post-process. Such a framework is presented

in Chapter 8 in the context of sand simulation. Therefore, a multi-resolution framework is presented, where the macroscopic behavior is efficiently simulated on a coarse-scale. In a second step, dispersion effects and interactions with complex boundaries are simulated on a significantly higher resolution. Visual detail is added which could not be resolved by the coarse simulation.

Finally, Chapter 9 concludes this thesis with a summary and possible directions for future work.

CHAPTER 2

SPH Fluid Simulation

SPH is a numerical interpolation method that will be used throughout this thesis, particularly for discretizing the fluid equations of motion. Therefore, this chapter presents the physical and numerical foundation of any SPH-based fluid solver. Section 2.1 gives an overview of fluid simulation techniques employed in computer graphics. In Section 2.2, the equations of motion for an incompressible fluid are derived. Subsequently, in Section 2.3, the SPH method is described. Finally, a simple SPH-based fluid solver is presented in Section 2.4.

2.1 Related Work

In computer graphics, the first water animations were presented in 1986 by Fournier and Reeves [FR86], and Peachy [Pea86]. These works model ocean surfaces using a single-valued height function. Wave trains are captured by relating the depth of water to the speed of waves. Such procedural models can track waves adequately over various terrains without solving the fluid differential equations. In [KM90], the model was extended to simulate reflection of waves and boundary conditions with changing topology. In general, height-field approximations are fast to compute and do approximate ocean surfaces on a coarse scale, but they fail to simulate many prominent effects found in a liquid. The reason is that these models ignore hydrodynamic field variables such as pressure.

More realism is achieved by employing a physical foundation. For fluids, this is provided by the Navier-Stokes equations, a set of partial differential equations which describes the motion of fluids. Chen and Lobo [CL95] headed into this direction by solving the Navier-Stokes equations in two dimensions and by mapping the corresponding pressures to the third dimension of the fluid's surface. This model can simulate three dimensional flow at interactive rates since the computational effort scales with the surface and not with the volume of the fluid. However, the depth information of the fluid body is lost which restricts the class of phenomena that can be solved by this method, e.g., the interaction with obstacles works only for objects whose cross-sectional area does not vary with depth.

Foster and Metaxas [FM96] introduced the first three dimensional fluid solver to computer graphics, namely the Marker-And-Cell (MAC) method proposed in [HW65]. In this method, the computational domain is divided using a Cartesian grid, where the velocity and pressure components are stored in a staggered fashion. According to the grid structure, the derivatives in the pressure solve are easily computed using finite differences, while staggering velocity and pressure components yields higher accuracy than regular grids. This technique

got very popular in computer graphics and is still at the heart of many fluid solvers, e.g., Hybrido [Nex11]. As a matter of course, many enhancements and variations have been proposed since then. An important improvement has been proposed by Stam [Sta99], who replaced the forward Euler integration with a backward particle trace. Thereby, the trajectory of hypothetical particles is used to advect the grid values. While the explicit integration scheme is only stable for sufficiently small time steps, the backward particle trace is unconditionally stable. However, the repetitive averaging of the advected quantity blurs sharp features like small vortices. This undesired effect, referred to as numerical dissipation, decreases with higher grid resolutions or smaller time steps. In [FSJ01], a solution is proposed that injects the lost energy back into the fluid using a forcing term. This technique is known as vorticity confinement [SU94].

A prominent strategy to counteract numerical dissipation is to transport the fluid properties with the particles as done in the Fluid-Implicit-Particle (FLIP) method. The FLIP method has been first described in [BR86] and introduced to computer graphics by Zhu and Bridson [ZB05]. In each time step, FLIP computes the equations of motions on a temporary background grid which is typically coarser sampled than the particle resolution. The velocity differences induced by the forces are then interpolated from the grid to the particles. The advection step is performed using the particles, i.e., particles are integrated in space and time according to the interpolated velocity. In the next time step, the grid velocities are interpolated back from the particles. As commonly employed in computer graphics, FLIP particles do not have a notion of their mass. On one hand this might cause volume compression due to accumulation of numerical errors, but on the other hand, such configurations do not destabilize the system.

A major challenge imposed by the Cartesian grid discretization is the accurate tracking of the fluid surface. The representation of the surface is not only important for rendering, but eminent for extrapolating unknown values at the free surface and at the interfaces with other materials. The state-of-the art to address this issue is to advect a level-set function with the flow, e.g., [FAMO99] and [Fed02]. The level-set method is a general framework for evolving interfaces which has been originally described by [OS88].

Another strategy to circumvent the challenges imposed by the grid discretization is to work exclusively on the particles. The most prominent representative of these mesh-free Lagrangian methods is SPH. The SPH method has been originally developed by Gingold and Monaghan [GM77] and Lucy [Luc77] for simulating astrophysical problems. SPH is not a specific solver, but an interpolation method for particle systems which is employed to compute field quantities and their spatial derivatives. Stam and Fiume introduced SPH to computer graphics for the simulation of gaseous phenomena and fire [SF95]. Later, Desbrun and Cani showed how to employ SPH to animate highly deformable bodies in [DC96].

The first liquid simulation based on SPH has been presented to the graphics community by Müller et al. in [MCG03]. This specific SPH solver has been designed for interactive simulations of compressible fluids. Inspired by the efficiency of SPH for compressible fluids, subsequent research, e.g., [BT07, SP09, MM13], has focused on practical, alternative formulations for incompressible fluids. This is important as incompressibility is a crucial property in order to obtain convincing simulations of water. This chapter focuses on a description of the fluid equations of motion and the SPH concept. At the end of this chapter, it is shown how SPH can be employed to obtain an approximative and simple discretization of the governing equations of a fluid using the example of [MCG03]. An in-depth discussion of alternative SPH solvers is provided in Chapter 3 which focuses on incompressibility.

2.2 Fluid Dynamics

2.2.1 The Material Derivative

In continuum mechanics, there exist two main approaches to describe a continuous volume, namely the *Eulerian* and the *Lagrangian* viewpoint. The Eulerian description measures material quantities at fixed spatial locations, i.e., the continuum is monitored from outside using a stationary coordinate system. In the Lagrangian viewpoint, the volume is represented by a discrete set of partial volumes referred to as parcels or particles. The material is described by watching the trajectory of each individual particle.

The material derivative $\frac{D}{Dt}$ is a differential operator designed to express the same phenomenon independent of the viewpoint. For both reference frames, it describes the temporal derivation of any property $\phi(\mathbf{x}, t)$ of a moving particle.

In the **Lagrangian** reference frame, the quantities are carried by the particles. For each instance in time t, we can access the position of a moving particle i at $\mathbf{x}_i(t)$. Assuming that the particles are advected by their velocity $\mathbf{v}_i$, the time rate of change of $\phi(\mathbf{x}_i(t), t)$ for particle i can be computed as

$$\frac{D\phi(\mathbf{x}_i(t), t)}{Dt} = \frac{d\phi(\mathbf{x}_i(t), t)}{dt} \approx \frac{\phi(\mathbf{x}_i(t + \Delta t), t) - \phi(\mathbf{x}_i(t), t)}{\Delta t}. \tag{2.1}$$

In order to compute the time rate of change of ϕ for a fluid particle at the time-invariant position $\mathbf{x}$ using the **Eulerian** frame, we have to compute the total derivative of $\phi(\mathbf{x}, t)$ with respect of time. This yields

$$\frac{D\phi(\mathbf{x}, t)}{Dt} \equiv \underbrace{\frac{\partial\phi(\mathbf{x}, t)}{\partial t}}_{\text{unsteady derivative}} + \underbrace{\mathbf{v}(\mathbf{x}, t) \cdot \nabla\phi(\mathbf{x}, t)}_{\text{convective derivative}}, \tag{2.2}$$

where $\mathbf{v} = (u, v, w)^T$ denotes the velocity of the fluid. $\frac{\partial\phi}{\partial t}$ measures the local time rate of change for a fixed point in space. This term is referred to as local or unsteady derivative as it is zero for steady flows. The convective term

$$\mathbf{v} \cdot \nabla\phi = u\frac{\partial\phi}{\partial x} + v\frac{\partial\phi}{\partial y} + w\frac{\partial\phi}{\partial z} \tag{2.3}$$

represents spatial changes in ϕ due to the flow around $\mathbf{x}$.

Although this thesis focuses on the Lagrangian approach SPH, the material derivative is employed in the following to derive the equation of motion for fluids. Thereby, a general description is obtained that can be mapped to both the Lagrangian and the Eulerian frame using (2.1) and (2.2), respectively.

2.2.2 Equations of Motion

We can think of the continuous volume as a set of partial volumes with mass m. Each parcel represents a piece of volume V at a position $\mathbf{x}$. Accordingly, the behavior of the continuum is determined by the dynamics of the parcels. This in turn means that it is sufficient to trace the motion of each parcel over time in order to compute the dynamics of the volume. For point masses with constant mass, the usual form of Newton's second law

$\mathbf{F} = m\mathbf{a}$ can be employed. However, parcels cannot be considered as point masses, but as volumetric elements. Thus, to derive the equation of motion for fluids, it is more appropriate to consider the momentum change of the volumes. This is expressed in words as

$$\begin{array}{c}\text{time rate of change of} \\ \text{momentum of a moving volume} \\ \text{element}\end{array} = \begin{array}{c}\text{sum of forces acting} \\ \text{on volume element}\end{array}. \tag{2.4}$$

Assuming constant properties throughout the volume, this relation can be derived from the actual version of Newton's second law

$$\mathbf{F} = D(m\mathbf{v})/Dt = V\frac{D(\rho\mathbf{v})}{Dt}, \tag{2.5}$$

where the volume $V = m/\rho$ is defined as mass m over density ρ. However, for infinitesimal volumes, the net force on the element would go to zero. Therefore, in fluid mechanics, the force per unit volume is of more relevance than the force. It is denoted by a lower case $\mathbf{f}$ and defined as

$$\mathbf{f} = \frac{\mathbf{F}}{V} = \frac{D(\rho\mathbf{v})}{Dt} \tag{2.6}$$

which states that the force per unit volume equals the time-rate of change of momentum per unit volume. If we remove the constant property assumption, we have to integrate the force and temporal change of momentum over the volume region $V(t)$ which yields

$$\int_{V(t)} \frac{\rho D\mathbf{v}}{Dt} dV = \int_{V(t)} \mathbf{f} dV. \tag{2.7}$$

Consequently, to integrate the system forward in time, we need to determine the total force $\mathbf{f}_i$ acting on each particle i. Basically, there are two kind of forces acting on a fluid particle, *body forces* and *surface forces*. Body forces are long-range forces acting on the entire volume region $V(t)$. Their origin is far away and their strength varies very slowly. Accordingly, they act uniformly on all particles. A typical body force is gravity

$$\mathbf{f}^g = \frac{m\mathbf{g}}{V} = \rho\mathbf{g}.$$

In the following, we represent all body forces with $\mathbf{f}^B$. Surface forces are short-range forces acting only on the surface $S(t)$ of the particle. They are imposed by surrounding fluid elements due to two sources: (i) the local pressure distribution, and (ii) shear and normal stress distributions. Pressure and normal stress are acting only in normal direction $\mathbf{n}$ to the element's surface, while shear stresses are acting tangentially to the surface. The net surface force can be written as $\mathbf{f}^S = \mathbf{T} \cdot \hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is a unit normal vector pointing outward to the surface on which $\mathbf{f}^S$ acts. $\mathbf{T}$ is a 3×3 matrix, carrying the physics represented by the vector $\mathbf{f}^S$. Plugging the partial forces into (2.7), we get

$$\begin{aligned}\int_{V(t)} \rho\frac{D\mathbf{v}}{Dt} dV &= \int_{V(t)} \mathbf{f}^B dV + \int_{S(t)} \mathbf{f}^S dS \\ &= \int_{V(t)} \mathbf{f}^B dV + \int_{S(t)} \mathbf{T} \cdot \hat{\mathbf{n}} dS.\end{aligned} \tag{2.8}$$

The surface integral can be converted to a volume integral by applying the Gauss theorem[1]

$$\int_{V(t)} \rho \frac{D\mathbf{v}}{Dt} dV = \int_{V(t)} \mathbf{f}^B dV + \int_{V(t)} \nabla \cdot \mathbf{T} dV. \tag{2.9}$$

This is already a very general description of the temporal change of momentum for any point in a fluid. We transform the integro-differential form of the equation into a pure differential form by shrinking the volume of the particle to an infinitesimal value such that $\rho \frac{D\mathbf{v}}{Dt}$, $\mathbf{f}^B$ and $\nabla \cdot \mathbf{T}$ are not varying throughout $V(t)$. Then, (2.9) becomes

$$\rho \frac{D\mathbf{v}}{Dt} = \mathbf{f}^B + \nabla \cdot \mathbf{T}. \tag{2.10}$$

Finally, we have to construct $\mathbf{T}$ such that it carries the physical information of the surface forces acting on a fluid element. The origin of these forces are pressure and viscosity. Pressure is a scalar value that is independent of orientation. As long as pressure does not vary spatially, the net force due to pressure is zero. However, if there is an imbalance, the net pressure force points from high pressure towards low pressure. Furthermore, pressure acts only in opposite direction of the fluid element's surface normal. Accordingly, we can split $\mathbf{T}$ into two tensors, one accounting for pressure p and one for viscosity denoted by τ

$$\mathbf{T} = -p\mathbf{I} + \tau, \tag{2.11}$$

where $\mathbf{I}$ is the identity matrix.

Viscosity describes the resistance of the fluid to deform. It has the physical effect of minimizing local velocity differences. In everyday terms, viscosity can be described as internal friction. Viscosity acts in normal and tangential direction, described by the viscous stress tensor τ as

$$\tau = \begin{bmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{bmatrix} = \mu \begin{bmatrix} \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x}\right) & \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \\ \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \left(\frac{\partial v}{\partial y} + \frac{\partial v}{\partial y}\right) & \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\ \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) & \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) & \left(\frac{\partial w}{\partial z} + \frac{\partial w}{\partial z}\right) \end{bmatrix}, \tag{2.12}$$

where μ is the dynamic viscosity coefficient. The right-hand side of (2.12) results from Newton's law of viscosity. As we have now specified $\mathbf{T}$, we can plug it into (2.10) to get

$$\rho \frac{D\mathbf{v}}{Dt} = \mathbf{f}^B - \nabla p + \nabla \cdot \tau. \tag{2.13}$$

Assuming incompressibility, $\nabla \cdot \tau$ can be written as $\mu \nabla^2 \mathbf{v}$. Here, ∇^2 denotes the *Laplacian operator* which is a measure for the distance of a quantity from the local average. Consequently, for an incompressible fluid element of infinitesimal size, the equation of motion can be written as

$$\rho \frac{D\mathbf{v}}{Dt} = \mathbf{f}^B - \nabla p + \mu \nabla^2 \mathbf{v}. \tag{2.14}$$

After rearranging and dividing the equation by ρ, we get

$$\underbrace{\mathbf{a}}_{\substack{\text{acceleration} \\ \text{of volume element}}} = \underbrace{-\frac{1}{\rho}\nabla p}_{\substack{\text{pressure forces} \\ \text{per unit mass}}} + \underbrace{\upsilon \nabla^2 \mathbf{v}}_{\substack{\text{viscosity forces} \\ \text{per unit mass}}} + \underbrace{\frac{\mathbf{F}^B}{m}}_{\substack{\text{body forces} \\ \text{per unit mass}}}, \tag{2.15}$$

[1] Gauss theorem: $\int_{V(t)} \nabla \cdot \mathbf{F} dV = \int_{S(t)} \mathbf{F} \cdot \hat{\mathbf{n}} dS$

where $\upsilon = \mu/\rho$ denotes the kinematic viscosity coefficient. For the simulation of low-viscous fluids like water, viscosity plays an important role in the formation of small-scale details like tiny water droplets. However, many numerical solvers introduce unintended damping due to smoothing operations. This damping is interpreted as artificial viscosity, while the explicit solution of the viscous stress is dropped.

Accordingly, for the animation of water, the pressure force is the most crucial force as it enforces incompressibility, i.e., constant volume. We will now derive a description of pressure for incompressible fluids. In an incompressible fluid, the volume is preserved over time. As the mass of a closed system does not change over time (*principle of mass*), the density $\rho = m/V$ is also constant. We start the derivation of the pressure description by expressing the conservation of mass mathematically as

$$\frac{dm}{dt} = \frac{d}{dt}\int_{V(t)} \rho dV = 0. \tag{2.16}$$

Please note that (2.16) also holds for compressible fluids as the middle term stems from $m = \rho V$ and not from the incompressibility condition.

Application of the general transport theorem[2] and the Gauss theorem to (2.16) yields

$$\int_{V(t)} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{v} \right) dV = 0. \tag{2.17}$$

Here, the choice of the volume region is arbitrary which implies that the integral is zero for any volume. Therefore, we can conclude that the integrand must be zero everywhere. Consequently, we can write the differential form as

$$\begin{aligned} 0 &= \frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{v} \\ &= \frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{v} \\ &= \frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v}. \end{aligned} \tag{2.18}$$

Recall that (2.18) just expresses that the mass is not changing over time. By employing the incompressibility assumption, namely that the density is constant, we yield the continuity equation for an incompressible flow:

$$\nabla \cdot \mathbf{v} = 0. \tag{2.19}$$

This equation expresses that mass and volume are conserved if and only if the velocity field is *divergence free* meaning that the divergence of the velocity field is zero. The only force that is maintaining this property is the pressure force.

In general, the best practice for a fluid solver is to solve the fluid equation in two steps. In the first step, the fluid is *advected*[3] to an intermediate state respecting all forces, but the pressure force. This step can be interpreted as a prediction step. In the second step, the pressure field is computed such that the intermediate velocity field gets divergence free. Accordingly, the second step corrects the velocity field and, hence, can be interpreted as a correction step. This method is often referred to as *splitting* as a complex equation is split up into its component parts and solved separately.

This section gave a general description of the equations that govern the behavior of incompressible fluids (2.15) and (2.19). The next sections present the Lagrangian SPH method which is employed in this thesis in order to solve the fluid equations.

[2] General transport theorem: $\frac{d}{dt}\int_{V(t)} \phi dV = \int_{V(t)} \frac{\partial \phi}{\partial t} dV + \int_{S(t)} \phi \mathbf{v} \cdot \hat{\mathbf{n}} dS$

[3] Advection means integration in space and time, sometimes also referred to as convection.

2.3 SPH Discretization

In SPH, the interpolation points are Lagrangian particles which move with the flow. Accordingly, the fluid volume is discretized into a finite set of volumetric particles with constant mass m and velocity $\mathbf{v}$. In each time step, the state of the total fluid volume is computed by updating the fluid equations for each particle. As depicted in the previous section, the fluid dynamics are described by spatial derivatives of physical quantities. Therefore, any numerical fluid solver has to approximate these derivatives. In Eulerian methods, the derivatives can be easily computed as finite differences using the grid vertices. In contrast, in SPH the particles are not spatially fixed and therefore another concept is required. This section first explains the interpolation concept SPH is based on and then describes how the spatial derivatives can be computed.

SPH employs the concept of integral representations of field variables. Based on this concept, any field variable $A(\mathbf{x})$ defined in a domain Ω can be expressed in terms of its values at a set of disordered points. Mathematically, $A(\mathbf{x})$ is replaced by an equivalent integral formulation as

$$A(\mathbf{x}) = \int_{\Omega} A(\mathbf{x}')\delta(\mathbf{x}-\mathbf{x}')d\mathbf{x}', \ \forall \mathbf{x}' \in \Omega \tag{2.20}$$

using the Dirac delta functional $\delta(\mathbf{x})$. The Dirac delta is defined as

$$\delta(x) = \begin{cases} \infty, \, x = 0 \\ 0, \quad x \neq 0 \end{cases}, \tag{2.21}$$

where $\delta(\mathbf{x}) = \delta(x)\delta(y)\delta(z)$. The functional is constrained to satisfy

$$\int_{-\infty}^{\infty} \delta(x)dx = 1. \tag{2.22}$$

Although the Dirac delta is rather a mathematical object than a true function, its property can be applied to approximate the value of a function at point $\mathbf{x}$ as

$$A(\mathbf{x}) = \lim_{\epsilon \to 0} \int_{-\infty}^{\infty} A(\mathbf{x}')\delta_{\epsilon}(\mathbf{x}-\mathbf{x}')d\mathbf{x}' \tag{2.23}$$

with

$$\delta_{\epsilon}(x) = \lim_{\epsilon \to 0} \begin{cases} 0, & x < -\epsilon/2 \\ 1/\epsilon, & -\epsilon/2 \leq x \leq \epsilon/2 \\ 0, & x > \epsilon/2 \end{cases}. \tag{2.24}$$

For numerical simulations, the Dirac delta functional needs to be replaced by a true function in (2.20). In SPH, the value $A(\mathbf{x})$ is approximated by replacing the Dirac delta with a kernel function $W(\mathbf{x}-\mathbf{x}', h)$, where h is the support of the function. The kernel function has to fulfill several requirements:

i) the integral must be normalized $\int_{-\infty}^{\infty} W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}' = 1$

ii) compact supportness, $W(\mathbf{x}-\mathbf{x}', h) = 0, \text{for } \|\mathbf{x}-\mathbf{x}'\| > h$

iii) it must converge to the Dirac delta functional for $h \to 0$

iv) it should be symmetric, $W(\mathbf{x}-\mathbf{x}', h) = W(\mathbf{x}'-\mathbf{x}, h)$

v) it should be non-negative, $W(\mathbf{x}-\mathbf{x}', h) \geq 0$.

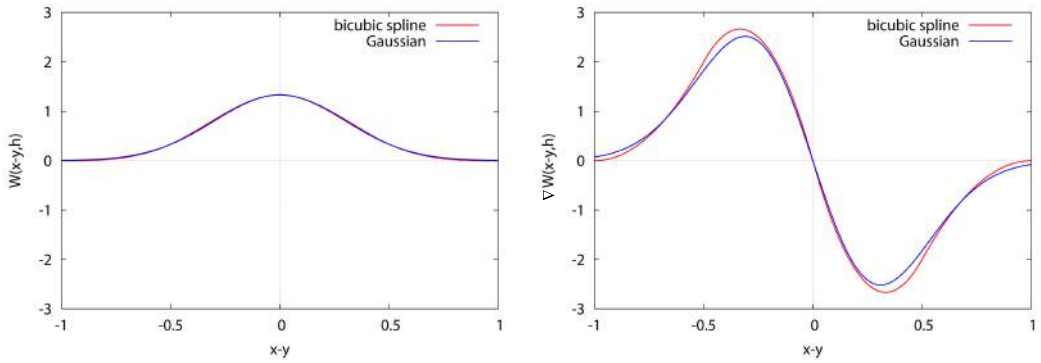

Figure 2.1: Comparison of the spline and Gaussian kernel in one dimension. The support radius h of the spline kernel (red) is set to 1.0. The Gaussian function (blue) is given for a standard deviation of 0.3. The kernel is symmetric (left), whereas the derivative (right) is anti-symmetric for any number of dimensions.

The *first golden rule of SPH* is that it is always best to assume the kernel to be a Gaussian [Mon92]. In fact, in the original work of Gingold and Monaghan [GM77], a Gaussian kernel is used. As in computer graphics, performance is crucial, spline functions [Mon92] are employed as they are more efficient to compute and are very similar to a Gaussian, see Figure 2.1.

Employing a suitable kernel W, (2.20) is approximated with

$$\langle A(\mathbf{x})\rangle = \int_{\Omega_h} A(\mathbf{x}')W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}', \tag{2.25}$$

where $\langle\cdot\rangle$ denotes the averaging operator. In the following, we will assume $\langle A(\mathbf{x})\rangle = A(\mathbf{x})$ as is usually done in the SPH literature. As the kernel is constrained to be compact, the integration over the entire domain Ω is localized as an integration over the support domain Ω_h of the smoothing function. This is illustrated in Figure 2.2.

Due to the symmetry, the kernel can be generally written as $W(\mathbf{x}-\mathbf{x}',h) = W\left(\frac{\|\mathbf{x}-\mathbf{x}'\|}{h}\right)$. Defining $q(\mathbf{x}) = \frac{\|\mathbf{x}\|}{h}$, the gradient of the kernel at $\mathbf{x}$ can be computed as

$$\nabla_{\mathbf{x}} W(q(\mathbf{x}-\mathbf{x}')) = \frac{\partial W}{\partial q}\frac{\mathbf{x}-\mathbf{x}'}{\|\mathbf{x}-\mathbf{x}'\|\,h}, \tag{2.26}$$

and the gradient at $\mathbf{x}'$ can be computed as $\nabla_{\mathbf{x}'} W(q(\mathbf{x}'-\mathbf{x})) = \frac{\partial W}{\partial q}\frac{\mathbf{x}'-\mathbf{x}}{\|\mathbf{x}-\mathbf{x}'\|h}$. Thus, the gradient of the kernel is always anti-symmetric $\nabla_{\mathbf{x}} W(q(\mathbf{x}-\mathbf{x}')) = -\nabla_{\mathbf{x}'} W(q(\mathbf{x}'-\mathbf{x}))$. This important property is employed in the following which shows how spatial derivatives can be computed in SPH. Detailed information on the design of the bicubic spline kernel and its derivatives are given in Appendix A.

2.3.1 SPH Derivatives

In SPH, any field variable A can be approximated with (2.25). The spatial derivative is obtained by computing the smoothed average of the gradient, i.e., substituting A with ∇A.

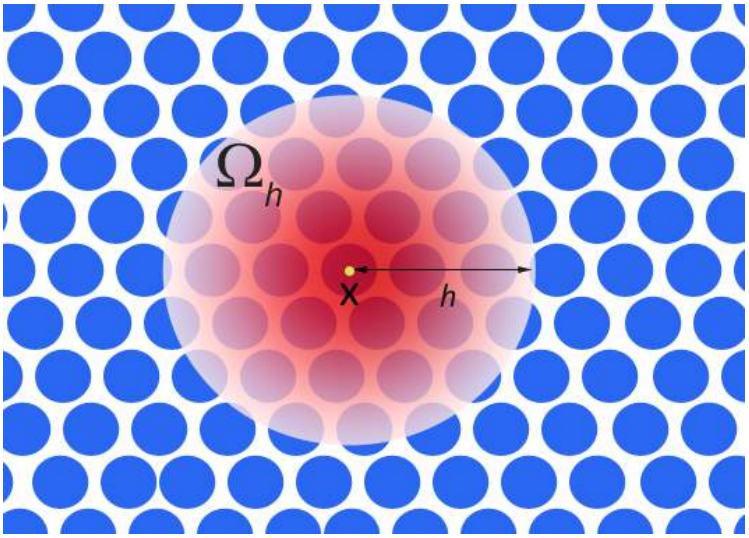

Figure 2.2: SPH interpolation at $\mathbf{x}$ over the local support domain Ω_h with radius h. The intensity of the weighting function $W(\mathbf{x}-\mathbf{x}', h)$ is color coded.

This yields

$$\nabla_{\mathbf{x}} A(\mathbf{x}) = \int_{\Omega_h} [\nabla_{\mathbf{x}'} A(\mathbf{x}')] W(\mathbf{x}-\mathbf{x}', h) d\mathbf{x}'. \tag{2.27}$$

Please note that if A is a scalar ∇A evaluates to a vector which is interpreted as the gradient. In three dimensions, ∇A is defined as $\nabla A = \frac{\partial A}{\partial x}\mathbf{e}_1 + \frac{\partial A}{\partial y}\mathbf{e}_2 + \frac{\partial A}{\partial z}\mathbf{e}_3$. If $\mathbf{A}$ is a vector, $\nabla \cdot \mathbf{A}$ denotes the divergence which is defined as $\nabla \cdot \mathbf{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$. Thus, $\nabla \cdot \mathbf{A}$ evaluates to a scalar value.

Applying the product rule, we can reformulate the integrand in (2.27) as

$$[\nabla_{\mathbf{x}'} A(\mathbf{x}')] W(\mathbf{x}-\mathbf{x}', h) = \nabla_{\mathbf{x}'} [A(\mathbf{x}') W(\mathbf{x}'-\mathbf{x}, h)] - A(\mathbf{x}') \nabla_{\mathbf{x}'} W(\mathbf{x}'-\mathbf{x}, h)$$

in order to yield

$$\nabla_{\mathbf{x}} A(\mathbf{x}) = \int_{\Omega_h} \nabla_{\mathbf{x}'} [A(\mathbf{x}') W(\mathbf{x}'-\mathbf{x}, h)] d\mathbf{x}' - \int_{\Omega_h} A(\mathbf{x}') \nabla_{\mathbf{x}'} W(\mathbf{x}'-\mathbf{x}, h) d\mathbf{x}'. \tag{2.28}$$

We can now convert the first integral on the right-hand side of (2.28) into an integral over the surface S_h of the support domain Ω_h by using the Gauss theorem as

$$\begin{aligned} \nabla_{\mathbf{x}} A(\mathbf{x}) &= \int_{S_h} A(\mathbf{x}') W(\mathbf{x}'-\mathbf{x}, h) \cdot \hat{\mathbf{n}} dS_h - \int_{\Omega_h} A(\mathbf{x}') \nabla_{\mathbf{x}'} W(\mathbf{x}'-\mathbf{x}, h) d\mathbf{x}' \\ &= \int_{S_h} A(\mathbf{x}') W(\mathbf{x}'-\mathbf{x}, h) \cdot \hat{\mathbf{n}} dS_h + \int_{\Omega_h} A(\mathbf{x}') \nabla_{\mathbf{x}} W(\mathbf{x}-\mathbf{x}', h) d\mathbf{x}', \end{aligned} \tag{2.29}$$

where we have exploited the anti-symmetric property of the kernel.

Since we have constrained the kernel function to have compact support, W is zero on the surface. Thus, the surface integral in (2.29) is zero, too. Consequently, we obtain the kernel approximation of the gradient as

$$\nabla_{\mathbf{x}} A(\mathbf{x}) = \int_{\Omega_h} A(\mathbf{x}') \nabla_{\mathbf{x}} W(\mathbf{x}-\mathbf{x}', h) d\mathbf{x}'. \tag{2.30}$$

Approximations of higher order derivatives are obtained similarly by applying integration by parts and the Gauss theorem to (2.25). Hence, the Laplacian $\nabla^2_{\mathbf{x}} A(\mathbf{x})$ can be written as

$$\nabla^2_{\mathbf{x}} A(\mathbf{x}) = \int_{\Omega_h} A(\mathbf{x}') \nabla^2_{\mathbf{x}} W(\mathbf{x} - \mathbf{x}', h) d\mathbf{x}'. \tag{2.31}$$

In a nutshell, the derivatives of any field value are easily determined by the derivatives of the smoothing function weighted by the function values in the support domain. This is a very important property for numerical fluid simulations as it allows a straightforward approximation of the spatial derivatives.

2.3.2 Particle Approximation

In SPH, the material is represented by volumetric particles which serve as interpolation points for solving the material's equations of motion. Therefore, each particle i is associated with a fixed mass m_i and represents a volume V_i which can be expressed in terms of density ρ and mass m as $V_i = m_i/\rho_i$. Here, and throughout the rest of this thesis, we employ the typical abbreviations used in the SPH literature where $A_i \equiv A(\mathbf{x}_i)$ denotes any field value at position $\mathbf{x}_i$. Based on this particle sampling, the continuous form of the SPH kernel interpolation (2.25) can be approximated as the *Riemann sum* over the finite set of sampling points $\mathbf{x}_j$ with

$$A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W_{ij}, \tag{2.32}$$

where $W_{ij} \equiv W(\mathbf{x}_i - \mathbf{x}_j, h)$. The first and second spatial derivatives of A_i are computed by approximating (2.30) as

$$\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}, \tag{2.33}$$

$$\nabla \cdot \nabla A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}, \tag{2.34}$$

where $\nabla W_{ij} \equiv \nabla_i W(\mathbf{x}_i - \mathbf{x}_j, h)$ and $\nabla W_{ji} \equiv \nabla_j W(\mathbf{x}_j - \mathbf{x}_i, h)$.

At this point, we have developed a first formalism to approximate the equations of motion for an incompressible fluid (2.15) and (2.19) in a numerical fashion. Recall that incompressible fluids are mainly described by the pressure force $\mathbf{f}_i^p = -\frac{m_i}{\rho_i} \nabla p_i$. According to (2.33) the pressure force can be formulated as

$$\mathbf{f}_i^p = -\sum_j \frac{m_j}{\rho_j} p_j \nabla W_{ij}. \tag{2.35}$$

However, this formulation does not preserve linear and angular momentum as the sum of forces is not zero in general. Furthermore, even if the pressure field is constant, $\mathbf{f}_i^p$ is not necessarily zero. In this case, the force is only zero if either the pressure values are zero or the neighborhood is sampled symmetrically. Monaghan [Mon92] proposes two alternative ways for computing the gradient in order to either preserve the momentum or to correctly approximate the derivative of constant functions. Thereby, ∇A_i is derived by placing the density inside as $\nabla (\rho_i A_i)$ or $\nabla \left(\frac{A_i}{\rho_i} \right)$ which either leads to a symmetric or anti-symmetric expression of the gradient. Monaghan refers to this technique as the *second golden rule of SPH* as it helps to improve the numerical accuracy of SPH simulations [Mon92].

Symmetric. Assuming constant mass and density throughout the volume, i.e., $m_i = m_j$ and $\rho_i = \rho_j$, a symmetric form of the spatial derivative can be derived. Therefore, ∇A_i is reformulated by placing the density inside as $\nabla(\rho_i A_i)$, applying the product rule and rearranging which reads

$$\nabla A_i = \frac{1}{\rho_i}\left[\nabla(\rho_i A_i) - A_i \nabla \rho_i\right]. \tag{2.36}$$

Approximating the right hand side of (2.36) with (2.33) yields

$$\begin{aligned}\nabla A_i &= \frac{1}{\rho_i}\left[\sum_j \frac{m_j}{\rho_j}\rho_j A_j \nabla W_{ij} - A_i \sum_j \frac{m_j}{\rho_j}\rho_j \nabla W_{ij}\right] \\ &= \frac{1}{\rho_i}\sum_j m_j (A_j - A_i)\nabla W_{ij}.\end{aligned} \tag{2.37}$$

According to this expression, the derivative vanishes for constant values of A independent of the sampling. However, employing (2.37) in order to compute the pressure force would not preserve linear and angular momentum. In order to show this, we write the contribution of particle j to the spatial derivative of the field value A_i as $\nabla A_{i\leftarrow j}$. Employing the assumption that mass and density are constant throughout the volume, we can derive that $\nabla A_{i\leftarrow j} = \nabla A_{j\leftarrow i}$ as

$$\begin{aligned}\nabla A_{i\leftarrow j} &= \frac{m_j}{\rho_i}(A_j - A_i)\nabla W_{ij} \\ &= \frac{m_i}{\rho_j}(A_i - A_j)\nabla W_{ji} \\ &= \nabla A_{j\leftarrow i}.\end{aligned}$$

Notice that even for variable values of m and ρ, $\nabla A_{j\leftarrow i}$ and $\nabla A_{i\leftarrow j}$ point into the same direction. Thus, the sum of forces is generally not zero if pressure forces are computed with (2.37) as $\mathbf{f}_i^p = -\frac{m_i}{\rho_i^2}\sum_j m_j (p_j - p_i)\nabla W_{ij}$.

Anti-symmetric. A formulation of the pressure force that preserves linear and angular momentum is derived by placing the density inside as $\nabla\left(\frac{A_i}{\rho_i}\right)$. This yields

$$\nabla\left(\frac{A_i}{\rho_i}\right) = \frac{\rho \nabla A_i - A_i \nabla \rho_i}{\rho_i^2} = \frac{\nabla A_i}{\rho_i} - \frac{A_i \nabla \rho_i}{\rho_i^2}.$$

Now, rearranging for a solution of ∇A_i reads

$$\nabla A_i = \rho_i\left(\nabla\left(\frac{A_i}{\rho_i}\right) + \frac{A_i \nabla \rho_i}{\rho_i^2}\right) \tag{2.38}$$

and together with the SPH approximation (2.33) of the derivative, we can write (2.38) as

$$\begin{aligned}\nabla A_i &= \rho_i\left[\sum_j \frac{m_j}{\rho_j}\frac{A_j}{\rho_j}\nabla W_{ij} + A_i \sum_j \frac{m_j}{\rho_j}\frac{\rho_j}{\rho_i^2}\nabla W_{ij}\right] \\ &= \rho_i \sum_j m_j\left(\frac{A_j}{\rho_j^2} + \frac{A_i}{\rho_i^2}\right)\nabla W_{ij}.\end{aligned} \tag{2.39}$$

Applying (2.39) for computing the pressure force guarantees that the pair-wise pressure-forces are anti-symmetric, i.e., $\mathbf{f}_{i\leftarrow j} = -\mathbf{f}_{j\leftarrow i}$. Hence, linear and angular momentum are preserved as the pressure forces sum up to zero.

Laplacian. The Laplacian can be symmetrized similarly by applying the product rule twice

$$\begin{aligned}\nabla^2(\rho_i A_i) &= A_i\left(\nabla^2\rho_i\right) + \rho_i\left(\nabla^2 A_i\right) + 2\nabla A_i \cdot \nabla\rho_i \\ \Leftrightarrow \nabla^2 A_i &= \frac{1}{\rho_i}\left[\nabla^2\left(\rho_i A_i\right) - A_i\left(\nabla^2\rho_i\right) - 2\nabla A_i \cdot \nabla\rho_i\right].\end{aligned}$$

The corresponding SPH expression is obtained by applying (2.34) which reads

$$\nabla^2 A_i = \frac{1}{\rho_i}\left[\nabla^2\left(\rho_i A_i\right) - A_i\left(\nabla^2\rho_i\right) - 2\nabla A_i \cdot \nabla\rho_i\right]. \tag{2.40}$$

However, approximating the second derivative directly with SPH is error prone at low resolutions and is very sensitive to particle disorder. This has been reported in [MFZ97]. Instead, Morris et al. [MFZ97] propose to approximate the second derivative with an hybrid expression obtained by combining the standard SPH first derivative with a finite difference approximation of a first derivative as

$$\nabla^2 A_i = 2\sum_j \frac{m_j}{\rho_j} A_{ij}\left(\frac{\mathbf{x}_{ij}\cdot\nabla W_{ij}}{\mathbf{x}_{ij}\cdot\mathbf{x}_{ij} + 0.01h^2}\right). \tag{2.41}$$

This section gave a general description of the SPH concept. By using the particle approximations of field values (2.32) and their derivatives (2.33), as well as by employing the numerical improvements (2.37) and (2.39), we can now discretize the fluid equations of motion with SPH.

2.4 Basic SPH Fluid Solver

In Section 2.2, we have derived the equation of motion for a moving fluid particle i which is given as

$$\frac{d\mathbf{v}_i}{dt} = -\frac{1}{\rho_i}\nabla p_i + \upsilon\nabla^2\mathbf{v}_i + \frac{\mathbf{F}_i^B}{m_i}. \tag{2.42}$$

In a typical simulation scenario, the only body force $\mathbf{F}_i^B$ is the gravity force $\mathbf{F}_i^{gravity} = m_i\mathbf{g}$. Accounting for gravity and multiplying (2.42) with the mass m_i, the net-force $\mathbf{F}_i$ acting on a particle can be computed as

$$\mathbf{F}_i(t) = -\frac{m_i}{\rho_i}\nabla p_i(t) + m_i\upsilon\nabla^2\mathbf{v}_i(t) + m_i\mathbf{g}. \tag{2.43}$$

In order to implement a simple SPH fluid solver, we just have to evaluate (2.43) in each simulation step. Then, we can integrate positions and velocities in time using, e.g., Euler-Cromer as

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t\frac{\mathbf{F}_i(t)}{m_i}, \tag{2.44}$$

$$\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \Delta t\mathbf{v}_i(t+\Delta t). \tag{2.45}$$

2.4.1 Forces

The net-force can be decomposed into pressure force $\mathbf{F}^{pressure}$, viscosity force $\mathbf{F}^{viscosity}$ and gravity force. A momentum-preserving approximation of the pressure force can be directly obtained by employing the anti-symmetric expression of the SPH gradient (2.39) which yields

$$\mathbf{F}_i^{pressure}(t) = -m_i \sum_j m_j \left(\frac{p_i(t)}{\rho_i^2(t)} + \frac{p_j(t)}{\rho_j^2(t)} \right) \nabla W_{ij}(t). \tag{2.46}$$

In order to compute the viscosity force, we have to approximate the second derivative of the velocity. Employing the numerically more robust approximation of the Laplacian (2.41), the viscosity can be computed as proposed in [MFZ97] with

$$\begin{aligned} \mathbf{F}_i^{viscosity} &= m_i \upsilon \nabla^2 \mathbf{v}_i(t) \\ &= m_i 2\upsilon \sum_i \frac{m_j}{\rho_j(t)} \mathbf{v}_{ij}(t) \left(\frac{\mathbf{x}_{ij}(t) \nabla W_{ij}(t)}{\mathbf{x}_{ij}^2(t) + 0.01h^2} \right). \end{aligned} \tag{2.47}$$

However, in computer graphics, the artificial viscosity force presented in [Mon92, BT07] is typically employed. The artificial viscosity force is defined as

$$\mathbf{F}_i^{viscosity} = \begin{cases} m_i \upsilon \sum_j \frac{2m_j}{\rho_j(t)+\rho_i(t)} \mathbf{v}_{ij}(t) \left(\frac{\mathbf{x}_{ij}(t) \nabla W_{ij}(t)}{\mathbf{x}_{ij}(t)^2 + 0.01h^2} \right) & \mathbf{v}_{ij}(t)\mathbf{x}_{ij}(t) < 0 \\ 0 & \mathbf{v}_{ij}(t)\mathbf{x}_{ij}(t) \geq 0 \end{cases} \tag{2.48}$$

with $\upsilon = \alpha h c_s$, where α is a user-given viscosity constant and c_s denotes the speed of sound. Note that the viscosity force described in (2.48) is a variant of (2.47) which is designed such that it dampens the relative velocity only for particles i and j that are approaching each other. The averaging of the volume in (2.48) ensures conservation of momentum.

2.4.2 Density and Pressure

In order to compute pressure and viscosity forces, density $\rho(t)$ and pressure values $p(t)$ have to be known. The density can be directly computed from the current particle configuration using the SPH approximation (2.32)

$$\rho_i(t) = \sum_j m_j W_{ij}(t). \tag{2.49}$$

What remains is the evaluation of the pressure field $p(t)$. There, exist various ways how to compute the pressure field. A straightforward approximation is obtained by locally relating pressure to the current density using the following equation of state (EOS)

$$p_i(t) = \frac{\kappa \rho_0}{\gamma} \left(\left(\frac{\rho_i(t)}{\rho_0} \right)^\gamma - 1 \right), \tag{2.50}$$

where κ and γ control the stiffness. The special case

$$p_i(t) = \kappa \left(\rho_i(t) - \rho_0 \right), \tag{2.51}$$

where $\gamma = 1$ is widely used in computer graphics, e.g., [MCG03].

Algorithm 1 SESPH

procedure NEIGHBORHOOD QUERY
  for all *particle* i **do**
    find neighbors
procedure COMPUTE DENSITY / PRESSURE
  for all *particle* i **do**
    compute $\rho_i(t) = \sum_j m_j W_{ij}(t)$
    compute $p_i(t)$ (2.50)
procedure COMPUTE FORCE
  for all *particle* i **do**
    compute $\mathbf{F}_i^{pressure}(t)$ (2.46)
    compute $\mathbf{F}_i^{viscosity}(t)$ (2.48)
    compute $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure}(t) + \mathbf{F}_i^{viscosity}(t) + \mathbf{g}$
procedure INTEGRATE
  for all *particle* i **do**
    compute $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i(t)/m_i$
    compute $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

2.4.3 Animation Loop

We now have all ingredients to build a first SPH fluid solver which is listed in Algorithm 1. In this basic solver just four loops over the particles are needed per simulation. According to the SPH scheme, scalar values and forces are interpolated locally. In order to guarantee an efficient implementation, particle neighbors should be queried and stored at the beginning of each simulation step. The second loop computes the density and pressure value field. After these values are computed, all forces can be computed in the third loop. The final loop integrates positions and velocities of the particles.

2.5 Discussion

In this chapter, the SPH method has been described and employed as a numerical method for computing the fluid equations of motion. This has been done using the example of the state-equation-based solver presented in [MCG03] which is still widely applied. The popularity of this solver stems from its simplicity. The pressure values can be directly computed without the need to solve a complex system of equations. However, there is no guarantee that the simulated fluid is incompressible as the continuity equation (2.19) is not taken into account. Using an EOS, pressure forces penalize compression, but do not guarantee an incompressible state at time $t + \Delta t$. In order to simulate weakly compressible fluids, a rather stiff EOS has to be used. Like for any penalty-based method this imposes a severe time step restriction limiting the overall performance. Chapter 3 focuses on the drawbacks of EOS-based methods and proposes a practical alternative.

CHAPTER 3

Incompressible SPH

Water is essentially incompressible. In everyday life, this property comes in handy as it helps to push water out of hoses or water fountains by only applying little pressure. The compressibility can be minimally affected by pressure and temperature, e.g., at a water depth of about 1500 meter, water compresses less than one percent although the weight of the water above applies a pressure that is 150 times higher than on the surface. Due to this property water can be even used for cutting and shaping materials like metals and ceramics by shooting highly pressurized water out of a nozzle at a speed of up to 1000 km/h.

Consequently, enforcing incompressibility is eminent to obtain realistic and visually pleasing simulations of water. However, maintaining incompressibility in SPH is numerically challenging and represents the computationally most expensive part of the simulation. This issue is addressed in this chapter which is organized as follows: After reviewing the related work, this chapter discusses the two strategies employed in the literature to model incompressible fluids with SPH, namely state-equation-based methods discussed in Section 3.2 and pressure-projection methods which are explained in Section 3.3. Subsequently, a new variant of the projection scheme is proposed which is derived by directly discretizing the continuous form of the continuity equation. The advantages of the proposed scheme with respect to performance, convergence and robustness can be assessed in Section 3.6, where various comparisons and convergence analyses are given. The practical relevance of the approach is illustrated by scenarios with up to 40 million SPH particles.

3.1 Related Work

In standard SPH (SESPH), an equation of state (EOS) is used to compute pressure that results in forces penalizing the current compression [Mon92]. Thereby, pressure is directly computed from locally evaluated density fluctuations weighted with a user-defined stiffness value. The fast and straightforward computation of pressure values makes SESPH well suited for efficient simulations of compressible fluids, e.g. [MCG03, KW06, APKG07], and has been also used to simulate multiple fluids [MSKG05, SP08], fluid-solid coupling [KAG+05, SSP07], melting solids [LAD08, BTT09] and fluid control [TKPR06]. However, for weakly compressible fluids (WCSPH), e.g. [Mon94, BT07], a rather stiff EOS has to be used in order to enforce a maximum compression of 1%. As for any penalty-based method, high stiffness imposes a severe time-step restriction, limiting the overall performance.

The performance of SESPH has been significantly improved by EOS-based predictor-corrector schemes, e.g., PCISPH [SP09] and local Poisson SPH [HLWW12]. In these approaches, pressure forces are modeled as constraint forces that resolve compression induced by non-pressure

forces. The respective pressure values are computed by iteratively predicting and correcting the particle positions based on an EOS. The approaches still rely on local information, but in contrast to SESPH, the EOS does not contain a user-defined stiffness parameter. PCISPH and local Poisson SPH handle time steps that are up to two orders of magnitude larger compared to WCSPH, while the overall speed-up with respect to WCSPH can be up to 55 or 23, respectively [SP09, HLWW12].

As an alternative to EOS approaches with locally computed penalty forces, projection schemes, also referred to as splitting [Bri08], can be used to compute the pressure field in SPH. First, intermediate velocities are predicted without considering the pressure forces. Then, a PPE is solved to compute pressure such that the resulting pressure forces correct the intermediate velocities to a divergence-free state. This is a standard technique in grid-based approaches, e.g. [FF01, Bri08, CM11]. In Lagrangian approaches, projection schemes can be distinguished with respect to the source term in the Poisson formulation. Here, either the divergence of the intermediate velocity field, e.g. [CR99, PTB+03], the compression due to the intermediate velocity field, e.g. [SL03, KGS09], or a combination of both, e.g. [HA07, LTKF08], are used. As incompressibility is an important ingredient for realistic fluid animations, the compression formulation seems to be preferable in SPH. As discussed for SPH in [CR99] and for the *Moving Particle Semi-Implicit* method (MPS) in [PTB+03], only using the divergence term tends to result in perceivable compression which is caused by the accumulation of numerical errors. Another issue is the SPH discretization of the Laplace operator with second-order kernel derivatives which is known to be sensitive to the sampling. This is discussed in, e.g. [MFZ97, CR99], where useful approximations are proposed to improve the convergence of the employed iterative solver.

In order to avoid the SPH discretization of the second derivative of the pressure in the PPE, some authors propose hybrid approaches that use an additional grid for the pressure computation [LKO05, LTKF08, RWT11]. Hybrid approaches compute the pressure field on the grid at a different, typically lower resolution. After transferring the pressure values from the grid to the particles, particle pressure values can be refined, e.g., using an EOS as in [RWT11].

SPH projection schemes, also referred to as incompressible SPH (ISPH) methods, are currently considered impractical in the context of computer graphics. As stated in, e.g. [CR99, ESE07], the performance of ISPH does not scale well with the problem domain which is particularly an issue for large-scale scenarios. This point is addressed in this chapter, where a new discretization of the PPE is proposed that significantly improves the convergence of the solver and the stability of the time-integration scheme. This results in a significant speedup not only compared to previous ISPH approaches, but more importantly compared to PCISPH, the current state-of-the-art. Furthermore, the proposed projection scheme scales well with the simulation domain. This is demonstrated by a number of large-scale scenarios.

3.2 EOS-based Methods

The SESPH solver described in Section 2.4 uses an EOS to compute pressure from locally evaluated compression, resulting in forces penalizing the current compression [Mon92]. The straightforward computation of pressure values makes SESPH well suited for efficient simulations of compressible fluids, e.g. [MCG03, APKG07]. However, in order to obtain weakly compressible fluids (WCSPH), e.g. [Mon94, BT07], a rather stiff EOS has to be used which

is defined as

$$p_i = \frac{\rho_0 c_s^2}{7}\left(\left(\frac{\rho_i}{\rho_0}\right)^7 - 1\right). \tag{3.1}$$

According to our experience, c_s should be set to 80 in order to maintain a volume compression of not more than 1%. However, as described in [BT07], c_s might be set to a smaller value if the depth of fluid volume is not too deep.

In SESPH, compression is locally penalized, but there is no guarantee that the compression is resolved at the next time step. In case that the time step is set too large, pressure forces might even amplify the compression which in turn causes the simulation to blow up. In fact, the stiffness required for simulations of incompressible fluids imposes a severe restriction of the time step which limits the overall performance significantly.

3.2.1 Predictive-corrective EOS Solver

The time step restriction of SESPH can be relaxed by applying the EOS in a predictive-corrective fashion, e.g., PCISPH [SP09] and local Poisson SPH [HLWW12]. Thereby, pressure forces are modeled as constraint forces that resolve compression induced by non-pressure forces. The respective pressure values are computed by iteratively predicting and correcting the particle positions based on an EOS. The iteration stops when the density error is below a user-defined threshold. Thereby, an incompressible state at time $t + \Delta t$ is guaranteed which positively affects the stability of the simulation. The following describes the concept of the solver in more detail.

Predictive-corrective EOS solvers relate pressure not to the current density deviation at time t, but to the predicted deviation at time $t + \Delta t$. The density of each particle is iteratively predicted as

$$\rho_i^l(t + \Delta t) = \sum_j m_j W\left(\mathbf{x}_i^l(t + \Delta t) - \mathbf{x}_j^l(t + \Delta t), h\right), \tag{3.2}$$

where l denotes the index of iteration. The predicted positions $\mathbf{x}_i^l$ are computed with

$$\mathbf{x}_i^l(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t) + \Delta t^2 \frac{\mathbf{F}_i^{adv}(t) + \mathbf{F}_i^p(t)}{m_i}. \tag{3.3}$$

Here, $\mathbf{F}_i^p(t)$ is the pressure force and $\mathbf{F}_i^{adv}(t)$ denotes non-pressure forces such as the gravity force and the viscosity force. While $\mathbf{F}_i^p(t)$ is a constrained force which uses implicit information, $\mathbf{F}_i^{adv}(t)$ uses exclusively information available at time t. This means $\mathbf{F}_i^{adv}(t)$ does not change during the iterations and needs to be evaluated only once.

Let us write the unconstrained position of each particle as $\mathbf{x}_i^{adv} = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t) + \Delta t^2 \frac{\mathbf{F}_i^{adv}(t)}{m_i}$. Employing Monaghan's momentum preserving pressure force (2.46), we can write the position change of particle i induced by the pressure force as

$$\begin{aligned}\mathbf{d}_i^l(t) &= \Delta t^2 \frac{\mathbf{F}_i^p(t)}{m_i} \\ &= \Delta t^2 \sum_j -m_j \left(\frac{p_i(t)^{l-1}}{\rho_i(t)^2} + \frac{p_j(t)^{l-1}}{\rho_j(t)^2}\right) \nabla W_{ij}(t).\end{aligned}$$

Algorithm 2 Predictive-corrective EOS solver. l indicates the iteration.

procedure FIND NEIGHBORS
  for all particle i **do**
    find neighbors $N_i(t)$
procedure PREDICT ADVECTION
  for all particle i **do**
    compute forces $\mathbf{F}_i^{adv}(t)$
    set pressure $p_i(t) = 0$
procedure PRESSURE SOLVE
  $l = 1$
  while $\rho_{avg}^l < \eta$ **do**
    for all *particle i* **do**
      compute $\mathbf{d}_i^l(t)$
      predict position $\mathbf{x}_i^l(t + \Delta t)$
    for all *particle i* **do**
      compute $\rho_i^l(t + \Delta t)$ (3.2)
      $p_i^l(t) += \kappa_i \left(\rho_i^l(t + \Delta t) - \rho_0\right)$
    $l = l + 1$
procedure INTEGRATION
  update velocity $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i} + \frac{\mathbf{d}_i^l(t)}{\Delta t}$
  update position $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

With these definitions, (3.3) reads

$$\mathbf{x}_i^l(t + \Delta t) = \mathbf{x}_i^{adv} + \Delta t^2 \sum_j -m_j \left(\frac{p_i(t)^{l-1}}{\rho_i(t)^2} + \frac{p_j(t)^{l-1}}{\rho_j(t)^2} \right) \nabla W_{ij}(t). \tag{3.4}$$

Employing an EOS, the pressure values are computed with

$$p_i^l(t) = \sum_l \kappa_i \left(\rho_i^l(t + \Delta t) - \rho_0\right). \tag{3.5}$$

Algorithm 2 lists the instructions for the described predictive-corrective EOS solvers. Note that there are several degrees of freedom when implementing this algorithm, e.g., in the choice of κ, the design of the EOS or the density prediction. In computer graphics, two variants of this algorithm are proposed, namely PCISPH [SP09] and iterative local Poisson SPH (i-LSPH) [HLWW12]. These methods mainly differ in the computation of the stiffness value and the choice of the EOS. Algorithm 2 directly corresponds to PCISPH if κ is computed as

$$\kappa_i = \frac{-1}{\beta_i(-\sum_j \nabla W_{ij}^0 \cdot \sum_j \nabla W_{ij}^0 - \sum_j (\nabla W_{ij}^0 \cdot \nabla W_{ij}^0))}, \tag{3.6}$$

where $W_{ij}^0 = W(\mathbf{x}_{ij}^0)$ with $\mathbf{x}_i^0$ denoting the initial position of a prototype particle with a filled neighborhood j and β is defined as

$$\beta_i = 2 \left(\frac{m_i \cdot \Delta t}{\rho_0} \right)^2. \tag{3.7}$$

In i-LSPH, a quadratic variant of the EOS $p_i^l(t) = \sum_l \kappa \rho_i^l(t+\Delta t) \left(\frac{\rho_i^l(t+\Delta t)}{\rho_0} - 1 \right)$ is employed. The stiffness value is computed as $\kappa_i = 0.5 \frac{r_i^2}{\Delta t^2}$, where r_i denotes the volume radius of the particle.

Both approaches still rely on local information, but in contrast to SESPH and WCSPH, the EOS does not contain a user-defined stiffness parameter. Due to the predictive-corrective formulation, the incompressibility condition is enforced for the next time step. For this reason, these schemes can handle larger time steps than the penalty-based EOS implementations [MCG03, BT07]. E.g., for a tolerated compression of 1% up to 35 larger time steps could be handled by PCISPH compared to WCSPH, which results in an overall speed-up of up to 17 [SP09].

3.3 Pressure-projection Methods

In pressure projection methods, the pressure force acts also like a constraint force which can be directly derived from the momentum and the continuity equation, i.e., without using an EOS. As for predictive-corrective EOS solvers, the concept of splitting is employed, i.e., the particles are first advected with respect to all forces, but the pressure force. This step is referred to as the prediction step. As a result of the prediction step, intermediate velocities $\mathbf{v}_i^{adv}(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i}$ for each particle i are obtained. The pressure force $\mathbf{F}_i^p(t)$ is then formulated as a constraint force which satisfies the continuity equation for incompressible fluids $\nabla \cdot \mathbf{v}_i(t + \Delta t) = 0$. Thus, the pressure force corrects $\mathbf{v}_i^{adv}(t + \Delta t)$ as

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^{adv}(t + \Delta t) + \Delta t \frac{\mathbf{F}_i^p(t)}{m_i}. \tag{3.8}$$

Expressing the pressure force in terms of the negative pressure gradient as in (2.15) yields

$$\mathbf{v}_i^{adv}(t + \Delta t) - \frac{\Delta t}{\rho_i(t)} \nabla p_i(t) = \mathbf{v}_i(t + \Delta t). \tag{3.9}$$

Taking the divergence of both sides, applying the incompressibility condition $\nabla \cdot \mathbf{v}_i(t+\Delta t) = 0$ to the right-hand side and reformulating yields the so called pressure Poisson equation (PPE)

$$\nabla \cdot (\nabla p_i) = \frac{\rho_i(t)}{\Delta t} \nabla \cdot \mathbf{v}_i^{adv}(t + \Delta t). \tag{3.10}$$

The right-hand side, referred to as the source term of the PPE, can be well approximated with SPH using the sampling invariant form of the gradient (2.37). This yields

$$\begin{aligned} \nabla \cdot \mathbf{v}_i^{adv}(t + \Delta t) &= \frac{\rho_i(t)}{\rho_i(t)} \sum_j m_j \left(\mathbf{v}_j^{adv}(t + \Delta t) - \mathbf{v}_i^{adv}(t + \Delta t) \right) \nabla W_{ij}(t) \\ &= - \sum_j m_j \left(\mathbf{v}_i^{adv}(t + \Delta t) - \mathbf{v}_j^{adv}(t + \Delta t) \right) \nabla W_{ij}(t). \end{aligned} \tag{3.11}$$

However, approximating the Laplacian on the left-hand side of (3.10) with SPH is numerically challenging as second-order kernel derivatives are sensitive to the sampling and may lead to a loss of resolution [All12]. Instead, the approximation of the Laplacian (2.41) is commonly employed, e.g., [CR99, SL03], in order to discretize the left-hand side as

$$\nabla^2 p_i(t) = \frac{2}{\rho_0} \sum_j m \left(\frac{p_{ij}(t) \mathbf{x}_{ij}(t) \nabla W_{ij}(t)}{\mathbf{x}_{ij}(t) \cdot \mathbf{x}_{ij}(t) + \eta^2} \right), \tag{3.12}$$

where $\eta = 0.01h$ is included to keep the denominator non-zero. This form is obtained by combining the standard SPH first derivative with a finite difference approximation of a first derivative. Furthermore, a specific symmetrization is employed which builds on the assumption that $\rho_i = \rho_j = \rho_0$ and $m_i = m_j = m$.

However, only using the divergence term results in perceivable compression due to accumulation of numerical errors as discussed in [CR99] and [PTB+03]. Therefore, the density invariant scheme as proposed in, e.g., [SL03], is preferred. In the density invariant scheme, the right-hand side is replaced by a density constraint as

$$\nabla^2 p_i(t) = \frac{\rho_0 - \rho_i^l(t + \Delta t)}{\Delta t^2}, \tag{3.13}$$

where the predicted density $\rho_i^l(t + \Delta t)$ is re-evaluated in each iteration l of the solver using position projection as in (3.2).

The left-hand side can be written as $\nabla^2 p_i(t) = \sum_j a_{ij} p_j$, where the coefficients a_{ij} are computed as

$$a_{ij} = -\frac{2}{\rho_0} m \left(\frac{\mathbf{x}_{ij} \nabla W_{ij}}{\mathbf{x}_{ij}^2 + \eta^2} \right) \tag{3.14}$$

and a_{ii} is computed as

$$a_{ii} = -\sum_{j \neq i} a_{ij}. \tag{3.15}$$

For grid-based solvers, pressure projection represents the state-of-the-art for computing incompressible flows. However, for SPH, the approximations employed for computing the Laplacian (3.12) are responsible for low convergence rates, particularly for larger domains. This issue is addressed in the following.

3.4 Implicit Incompressible SPH

The proposed implicit incompressible SPH (IISPH) method is closely related to pressure projection methods. It significantly differs in the discretization of the PPE as assumptions, approximations or simplifications are avoided. The novel formulation is derived by combining the symmetric SPH pressure force and an SPH discretization of the continuity equation. In contrast to all previous projection schemes, the resulting system does consider the actual computation of the pressure force. This incorporation improves the convergence rate of the solver. Furthermore, we propose to compute the density deviation based on velocities instead of positions as this formulation improves the robustness of the time-integration scheme.

3.4.1 Derivation

IISPH is based on a semi-implicit form of the density prediction using the time rate of change of the density. The formulation is obtained by directly discretizing the continuity equation $\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}$ at time $t + \Delta t$ using a backward difference for the time derivative of the density $\frac{\rho_i(t+\Delta t) - \rho_i(t)}{\Delta t}$ and the SPH concept for the divergence of the velocity $\nabla \cdot \mathbf{v}_i = -\frac{1}{\rho_i} \sum_j m_j \mathbf{v}_{ij} \nabla W_{ij}$, which yields

$$\frac{\rho_i(t + \Delta t) - \rho_i(t)}{\Delta t} = \sum_j m_j \mathbf{v}_{ij}(t + \Delta t) \nabla W_{ij}(t). \tag{3.16}$$

This specific discretization introduces unknown relative velocities $\mathbf{v}_{ij}(t+\Delta t) = \mathbf{v}_i(t+\Delta t) - \mathbf{v}_j(t+\Delta t)$ that depend on unknown pressure forces at time t which are linear in unknown pressure values at time t.

Linear system

Using a semi-implicit Euler scheme for position and velocity update, the velocity in (3.16) can be rewritten as: $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)+\mathbf{F}_i^p(t)}{m_i}$ with unknown pressure forces $\mathbf{F}_i^p(t)$ and known non-pressure forces $\mathbf{F}_i^{adv}(t)$ as in Section 3.2.1. Following the projection concept, we consider intermediate (predicted) velocities $\mathbf{v}_i^{adv} = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i}$ which result in an intermediate density

$$\rho_i^{adv} = \rho_i(t) + \Delta t \sum_j m_j \mathbf{v}_{ij}^{adv} \nabla W_{ij}(t). \tag{3.17}$$

We now search for pressure forces to resolve the compression $\rho_0 - \rho_i^{adv}$:

$$\Delta t^2 \sum_j m_j \left(\frac{\mathbf{F}_i^p(t)}{m_i} - \frac{\mathbf{F}_j^p(t)}{m_j} \right) \nabla W_{ij}(t) = \rho_0 - \rho_i^{adv}. \tag{3.18}$$

Note, that (3.18) corresponds to (3.16) with $\rho_i(t+\Delta t) = \rho_0$. Using the symmetric pressure force (2.46) in (3.18), we get a linear system $\mathbf{A}(t)\mathbf{p}(t) = \mathbf{b}(t)$ with n equations for n unknown pressure values $\mathbf{p}(t) = \big(p_0(t), \ldots, p_{n-1}(t)\big)^{\mathrm{T}}$ and $\mathbf{b}(t)$ corresponding to the right-hand side of (3.18), i.e., $b_i(t) = \rho_0 - \rho_i^{adv}$. For each particle, we finally have an equation of the form

$$\sum_j a_{ij} p_j = b_i = \rho_0 - \rho_i^{adv}, \tag{3.19}$$

where we have skipped the time index t to improve the readability. This is also done in the following.

3.4.2 Properties

Equation (3.17) corresponds to the prediction step of the projection, while (3.18) is the PPE that is used for the pressure computation. The respective pressure forces are applied in the correction step of the projection scheme: $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i^{adv} + \Delta t \mathbf{F}_i^p(t)/m_i$. The source term states the density invariance condition which we prefer over the alternative divergence term. The divergence term tends to result in problematic compression as discussed in, e.g. [CR99, PTB+03]. Thus, our approach is closely related to SPH projection schemes, also referred to as incompressible SPH (ISPH), which employ density constraints, e.g., [SL03]. However, our formulation significantly improves the convergence of the solver and the stability of the time-integration scheme due to two novel aspects discussed in the following.

Discretization of the Laplacian. Our discretization of the PPE significantly differs from previous formulations that employ additional approximations for the Laplace operator, e.g., [CR99, SL03, HA07, KGS09]. All previous ISPH methods start with a continuous form of the PPE. Then, they discretize the Laplacian and the source term. The resulting system does not consider the pressure force that is finally derived from the pressure field. Thus, there is no distinguished form of the pressure force with a special relation to the computed

pressure field. In contrast to all previous ISPH approaches, our derivation considers the relation between pressure and pressure force. First, we do not start with a continuous PPE, but with the continuous continuity equation which is discretized. The main goal of this discretization is the introduction of $\mathbf{v}(t + \Delta t)$ which is expressed with the pressure force term used in the final velocity update. Thereby, we can finally apply the particular form of the pressure force that has been considered in the computation of the pressure field. This incorporation improves the convergence as shown in comparisons to ISPH, see Section 3.6.

Source term. In previous ISPH implementations and in predictive-corrective EOS solvers, $\rho_i(t + \Delta t)$ is computed based on predicted positions as

$$\rho_i(t + \Delta t) = \sum_j m_j W(\mathbf{x}_i^* - \mathbf{x}_j^*, h), \tag{3.20}$$

where $\mathbf{x}_i^*$ equals $\mathbf{x}_i^{adv}$ in ISPH and the predicted positions during iterations in predictive-corrective EOS solvers. However, solving (3.20) implies a re-computation of the neighborhood. In PCISPH, this significant overhead is avoided by only updating distances for the current neighborhood. This introduces an error which gets more significant with larger displacements $\Delta\mathbf{x}_i = \Delta t\mathbf{v}_i^{adv} + \Delta t^2\mathbf{F}_i^p/m_i$.

In contrast, IISPH predicts the density based on velocities (3.17). Known values $\nabla W_{ij}(t)$ can be preferred over unknown values $\nabla W_{ij}(t + \Delta t)$ without affecting the error order of the discretization. Thereby, we avoid the approximative update of the neighborhood. According to our observations, (3.17) tolerates significantly larger time steps and, thus, improves the robustness of the time-integration scheme.

In summary, our discretization of the Laplacian improves the convergence, while the source term allows for large timesteps. On the other hand, the system contains a significantly larger number of non-zero entries compared to previous projection schemes. As (3.19) contains a nested sum, the coefficients a_{ij} are non-zero for the neighbors j of particle i and for neighbors of neighbors of i. Typically, a particle has 30-40 neighbors [Mon05]. Nevertheless, the system can be solved very efficiently in a matrix-free way as presented in the following.

3.5 Solver

The system can be solved in various ways. For similar systems, grid approaches commonly apply SOR, e.g. [FM96], conjugate gradient, e.g. [FF01], or multigrid solvers, e.g. [MST10, CM11], while SPH approaches often employ conjugate gradient for PPEs, e.g. [CR99]. We have implemented and evaluated relaxed Jacobi and conjugate gradient to solve our formulation. Multigrid solvers have not been considered due to their involved setup for irregular samplings.

For the proposed system, the relaxed Jacobi solver is more practical than conjugate gradient. We therefore describe implementation details of the relaxed Jacobi solver first in Section 3.5.1, followed by a discussion of possible reasons for that outcome in Section 3.5.2.

3.5.1 Relaxed Jacobi

Employing relaxed Jacobi, we iteratively solve (3.19) for the individual pressure values p_i as

$$p_i^{l+1} = (1 - \omega)p_i^l + \omega\frac{\rho_0 - \rho_i^{adv} - \sum_{j\neq i} a_{ij}p_j^l}{a_{ii}}, \tag{3.21}$$

where l denotes the iteration index and ω is called the relaxation factor.

In order to compute (3.21), we need to determine a_{ii} and $\sum_{j \neq i} a_{ij} p_j^l$ which can be efficiently computed. For extracting the coefficients, the displacement caused by the pressure force is rewritten as

$$\begin{aligned} \Delta t^2 \frac{\mathbf{F}_i^p}{m_i} &= -\Delta t^2 \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} \\ &= \underbrace{\Big(- \Delta t^2 \sum_j \frac{m_j}{\rho_i^2} \nabla W_{ij} \Big)}_{\mathbf{d}_{ii}} p_i + \sum_j \underbrace{-\Delta t^2 \frac{m_j}{\rho_j^2} \nabla W_{ij}}_{\mathbf{d}_{ij}} p_j, \end{aligned} \tag{3.22}$$

where $\mathbf{d}_{ii} p_i$ denotes the displacement of i due to pressure values p_i and $\mathbf{d}_{ij} p_j$ expresses the movement caused by the pressure value p_j of neighboring particle j. Plugging (3.22) into (3.18) and denoting the neighbors of j with k yields

$$\begin{aligned} &\rho_0 - \rho_i^{adv} = \\ &\sum_j m_j \Big(\mathbf{d}_{ii} p_i + \sum_j \mathbf{d}_{ij} p_j - \mathbf{d}_{jj} p_j - \sum_k \mathbf{d}_{jk} p_k \Big) \nabla W_{ij}. \end{aligned} \tag{3.23}$$

Note that $\sum_k \mathbf{d}_{jk} p_k$ includes pressure values p_i since i and j are neighbors. In order to separate p_i in this sum, we write

$$\sum_k \mathbf{d}_{jk} p_k = \sum_{k \neq i} \mathbf{d}_{jk} p_k + \mathbf{d}_{ji} p_i. \tag{3.24}$$

Taking these considerations into account, the right-hand side of (3.23) can be split up into parts that contain p_i values and other parts that contain pressures p_j and p_k as

$$\begin{aligned} \rho_0 - \rho_i^{adv} = p_i \sum_j m_j \left(\mathbf{d}_{ii} - \mathbf{d}_{ji} \right) \nabla W_{ij} \\ + \sum_j m_j \Big(\sum_j \mathbf{d}_{ij} p_j - \mathbf{d}_{jj} p_j - \sum_{k \neq i} \mathbf{d}_{jk} p_k \Big) \nabla W_{ij}. \end{aligned}$$

Now, we can compute the coefficients a_{ii} as

$$a_{ii} = \sum_j m_j \left(\mathbf{d}_{ii} - \mathbf{d}_{ji} \right) \nabla W_{ij}, \tag{3.25}$$

and evaluate the pressure p_i^{l+1} with

$$\begin{aligned} p_i^{l+1} = (1 - \omega) p_i^l + \omega \frac{1}{a_{ii}} \Bigg(\rho_0 - \rho_i^{adv} \\ - \sum_j m_j \Big(\sum_j \mathbf{d}_{ij} p_j^l - \mathbf{d}_{jj} p_j^l - \sum_{k \neq i} \mathbf{d}_{jk} p_k^l \Big) \nabla W_{ij} \Bigg). \end{aligned} \tag{3.26}$$

Algorithm 3 IISPH using relaxed Jacobi. l indicates the iteration.

procedure PREDICT ADVECTION
  for all *particle* i **do**
    compute $\rho_i(t) = \sum_j m_j W_{ij}(t)$
    predict $\mathbf{v}_i^{adv} = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i}$
    $\mathbf{d}_{ii} = \Delta t^2 \sum_j -\frac{m_j}{\rho_i^2} \nabla W_{ij}(t)$
  for all *particle* i **do**
    $\rho_i^{adv} = \rho_i(t) + \Delta t \sum_j m_j \left(\mathbf{v}_{ij}^{adv}\right) \nabla W_{ij}(t)$
    $p_i^0 = 0.5\ p_i(t - \Delta t)$
    compute a_{ii} (3.25)
procedure PRESSURE SOLVE
  $l = 0$
  while $\rho_{avg}^l - \rho_0 > \eta \vee l < 2$ **do**
    for all *particle* i **do**
      $\sum_j \mathbf{d}_{ij} p_j^l = \Delta t^2 \sum_j -\frac{m_j}{\rho_j^2(t)} p_j^l \nabla W_{ij}(t)$
    for all *particle* i **do**
      compute p_i^{l+1} (3.26)
      $p_i(t) = p_i^{l+1}$
    $l = l + 1$
procedure INTEGRATION
  for all *particle* i **do**
    $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^{adv} + \Delta t \mathbf{F}_i^p(t)/m_i$
    $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

Implementation

In our implementation, $\mathbf{d}_{ii}$ and a_{ii} are precomputed and stored. The coefficients $\mathbf{d}_{ij}$ are computed to get a_{ii}, but not stored. In each iteration, two passes over the particles are required to update p_i^{l+1}. The first pass computes and stores $\sum_j \mathbf{d}_{ij} p_j^l$. The second pass computes p_i^{l+1} using the stored $\sum_j \mathbf{d}_{ij} p_j^l$ and $\mathbf{d}_{jj}$. The sum $\sum_{k \neq i} \mathbf{d}_{jk} p_k^l$ is computed as $\sum_k \mathbf{d}_{jk} p_k^l - \mathbf{d}_{ji} p_i^l$, where the term $\sum_k \mathbf{d}_{jk} p_k^l$ can be accessed at the particle, while $\mathbf{d}_{ji}$ is computed. Thus, an explicit computation of the non-diagonal elements a_{ij} is avoided. Algorithm 3 summarizes the simulation update using the proposed relaxed Jacobi pressure solve.

IISPH performs two loops over all particles per iteration. As these two loops do not contain data dependencies, the method is well suited for parallel architectures. While an equation for a particle contains up to 40^2 non-zero coefficients, the implementation only stores seven scalar values per particle, namely a_{ii}, $\mathbf{d}_{ii}$ and $\sum_j \mathbf{d}_{ij} p_j^l$.

On multi-core CPU architectures it pays off to store pairwise distance information when querying the neighbors in order to accelerate kernel and kernel gradient computations (see analysis in Chapter 4). In the iterative pressure computation of IISPH only kernel gradients are required, but no kernel weights. As the kernel gradient can be decomposed into the product of a scalar value w_{ij} and the vector $\mathbf{x}_{ij}$ as $\nabla W_{ij} = w_{ij} \mathbf{x}_{ij}$, the scaling of the kernel gradients w_{ij} can be precomputed per pair. The pairwise storage of the scalar values does not increase the memory consumption compared to a typical SPH implementation which precomputes pairwise distances. In contrast, for PCISPH, pairwise kernel values and kernel

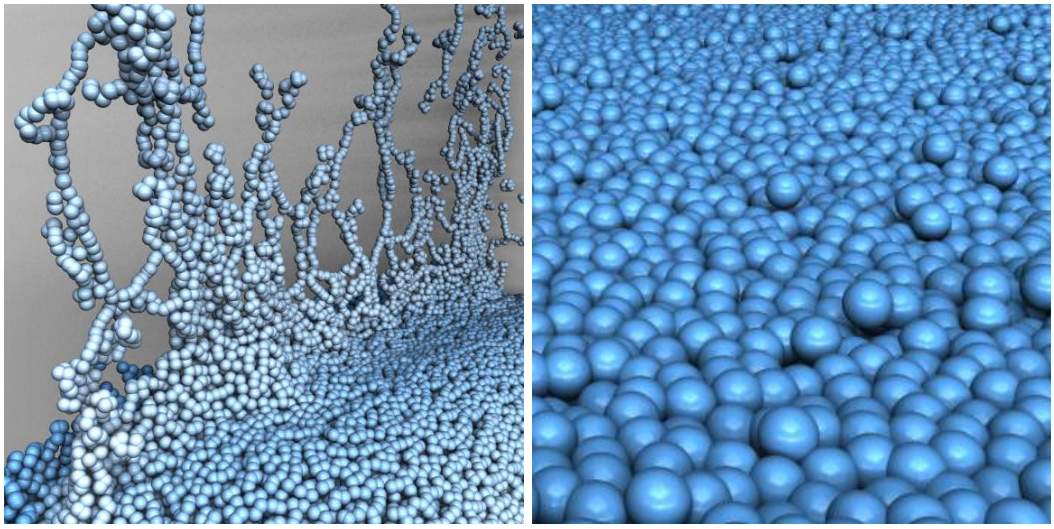

Figure 3.1: Exaggerated surface tension by attracting pressure forces (left). Distorted alignment of surface particles (right) caused by clamping $b_i = min(0, \rho_0 - \rho_i^{adv})$. Particles are color coded with respect to velocity.

gradients are employed per iteration. In this case, pre-computing distance values is most efficient with respect to the performance.

We observed an optimal convergence for the relaxation factor $\omega = 0.5$ in all settings. The convergence is also optimized by initializing $p_i^0 = 0.5\, p_i(t - \Delta t)$. While grid-based projection schemes commonly initialize $p_i^0 = 0$, e.g. [Bri08], using $p_i^0 = p_i(t - \Delta t)$ would be intuitive in our Lagrangian approach. In our experiments, however, we observed that the coefficient 0.5 generally provides a close to optimal convergence. The right-hand side of (3.26) can be rearranged to compute the density at $t + \Delta t$ including pressure forces. Thus, the compression of the fluid can be predicted in each iteration. By terminating the iterations for a compression below a pre-defined value η, the user can control the compression.

By enforcing $\rho_i(t + \Delta t) = \rho_0$, pressure forces induce a positive change of density for particles with a predicted density smaller than the rest density, i.e., $\rho_i^{adv} < \rho_0$. In this case, pressure forces act as attraction forces [SB12]. Although this might be interpreted as surface tension, we consider the induced cohesive effect as too exaggerated, e.g., splashes are noticeably absorbed (Figure 3.1, left). In EOS solvers, attracting pressure forces are prohibited by clamping negative pressures to zero. We adopt the same concept and clamp negative pressures in each iteration.

Finally, in contrast to hybrid particle-mesh solvers, the proposed method builds purely on SPH. Thus, it can be integrated into sophisticated SPH frameworks, e.g. [SG11], as demonstrated in Figure 3.2.

3.5.2 Conjugate Gradient

We have implemented conjugate gradient (CG) with a diagonal pre-conditioner. In each iteration, CG updates the individual pressure values p_i as $p_i^{l+1} = p_i^l + \alpha^l d_i^l$ where α^l denotes the step-width and $\mathbf{d}^l$ a direction. The new direction is updated with respect to the residual $r_i^{l+1} = r_i^l + \alpha^l \sum_j a_{ij} d_i^l$ as $d_i^{l+1} = r_i^{l+1} + \beta^l d_i^l$. β is for example computed as $\beta^l = \frac{\mathbf{r}^{l+1}\mathbf{r}^{l+1}}{\mathbf{r}^l\mathbf{r}^l}$.

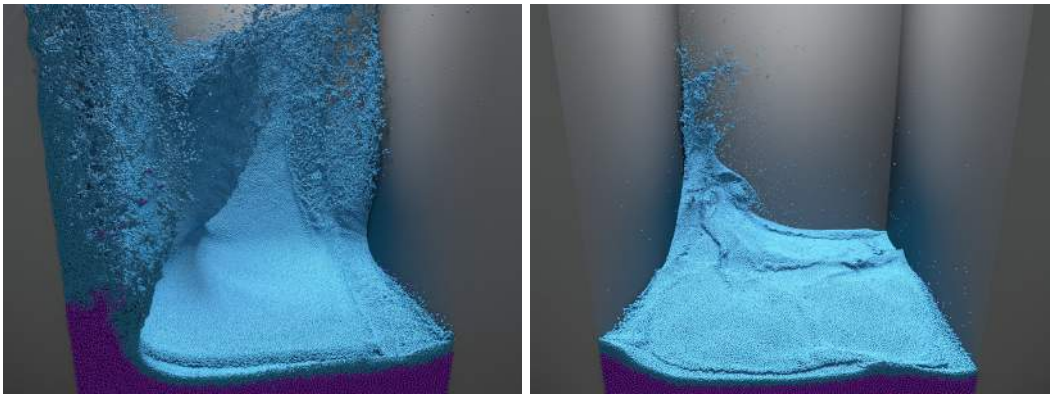

Figure 3.2: A two-scale simulation with IISPH. The radii of the pink low-resolution and the blue high-resolution particles are 0.004m and 0.002m, respectively.

Like for relaxed Jacobi, it is useful to not explicitly represent $\mathbf{A}$, but compute $\sum_j a_{ij} d_i^l$ on-the-fly. Therefore, one additional pass is required to compute a three dimensional vector $\mathbf{a}_i^l$ for each particle as

$$\mathbf{a}_i^l = -\sum_j m_j \left(\frac{d_i^l}{\rho_i^2} + \frac{d_j^l}{\rho_j^2} \right) \nabla W_{ij}. \tag{3.27}$$

Taking (3.27), $\sum_j a_{ij} d_i^l$ can be computed with $\sum_j a_{ij} d_i^l = \Delta t^2 \sum_j m_j \left(\mathbf{a}_i^l - \mathbf{a}_j^l \right) \nabla W_{ij}$. Algorithm 4 lists the pressure projection with conjugate gradient. Although CG shows promising convergence rates, we experienced three issues which are highlighted in the following.

Algorithm 4 Pressure solve with conjugate gradient. l indicates the iteration. Advection and integration are simplified.

procedure PREDICT ADVECTION
procedure PRESSURE SOLVE
  $l = 1$
  while $r_{avg}^l < \eta$ **do**
    set α^l, β^l to zero
    for all *particle* i **do**
      compute $\mathbf{a}_i$ (3.27)
    for all *particle* i **do**
      $\alpha_{nom}^l += d_i^l r_i^l$
      $\alpha_{denom}^l += d_i^l \Delta t^2 \sum_j m_j \left(\mathbf{a}_i^l - \mathbf{a}_i^l \right) \nabla W_{ij}(t)$
    $\alpha^l = \alpha_{nom}^l / \alpha_{denom}^l$
    for all *particle* i **do**
      compute $p_i^{l+1} = p_i^l + \alpha^l d_i^l$
      compute r_i^{l+1}
      $\beta_{nom}^l += r_i^{l+1} r_i^{l+1}$; $\beta_{denom}^l += r_i^l r_i^l$
    $\beta^l = \beta_{nom}^l / \beta_{denom}^l$
    for all *particle* i **do**
      compute $d_i^{l+1} = r_i^{l+1} + \beta^l d_i^l$
    $l = l + 1$
procedure INTEGRATION

Symmetry. The coefficient matrix is not symmetric as each a_{ij} is scaled by m_i/ρ_i^2. Symmetry can be enforced by assuming that $\rho_i = \rho_j = \rho_0$ and $m_i = m_j$ for all particles. While this symmetrization works for uniform masses, it is invalid for settings with non-uniform masses such as adaptively sampled SPH, e.g. [APKG07], or multi-phase simulations, e.g. [SP08]. Note that non-symmetry is not an issue for relaxed Jacobi.

Negative Pressure. In our relaxed Jacobi implementation, negative pressures are clamped to eliminate exaggerated cohesion effects. For CG, however, clamping in between the iterations leads to invalid states. We also observed instabilities in case of any change in the final pressure field, such as clamping of negative pressure values or disregarding pairwise-attracting pressure forces. Intuitively, we could disallow a positive change of density due to pressure by clamping b_i to negative values with $b_i = \min\left(0, \rho_0 - \rho_i^{adv}\right)$. Unfortunately, this adaptation causes implausible alignments of single particles at the fluid surface for CG and relaxed Jacobi as demonstrated in the right-hand side of Figure 3.1. This issue might be addressed by incorporating ghost particles [SB12] in order to improve the density evaluation ρ_i at time t near the free surface. However, it has to be investigated if the proposed re-sampling of ghost particles might affect the stability of incompressible SPH implementations as the ghost-particle method has not yet been incorporated into an incompressible SPH solver.

Performance. Employing Jacobi preconditioning by extracting the diagonal elements a_{ii} as explained in Section 3.5.1, the convergence rate of CG is notably higher than for relaxed Jacobi. However, the CG implementation requires two additional loops over the particles per iteration and, thus, almost doubles the costs per iteration compared to the relaxed Jacobi implementation.

3.6 Results

The solvers, namely PCISPH, ISPH and IISPH, have been incorporated into an SPH framework which employs the bicubic spline kernel and the viscous force (2.48) presented in [Mon05]. Surface tension is modeled with [BT07]. Rigid-body coupling is realized with [AIS+12]. Chapter 5 shows how to incorporate [AIS+12] in IISPH. Neighborhood search and fluid update are parallelized using the techniques described in Chapter 4. The fluid surface is reconstructed with [SSP07, AIAT12]. Performance measurements are given for a 16-core 3.46 GHz Intel i7 with 64 GB RAM. Images were rendered with mental ray [NVI11].

3.6.1 Convergence Criterion

Realistic SPH simulations require low density errors in order to avoid perceivable volume changes, i.e., oscillations of the free surface. While the original PCISPH method considered the maximum density error with a threshold of 1%, we recommend to maintain an average density error of less than 0.1%. Although average and maximum density error are closely related, their ratio varies throughout a simulation. I.e., if a maximum density error is preserved, the average density error and therefore the fluid volume might change over time. Such variations of the average density error result in oscillations of the pressure field and in perceivable simulation artifacts such as jumping of the free surface. This issue is particularly perceivable in simulations with growing water depth as illustrated in Figure 3.3. In this scenario, the obtained average compression varied between 0.3% and 0.6% when tolerating a maximum error of 1%. We found that these artifacts can be avoided by considering the estimated average density error with a threshold of 0.1% as the convergence criterion. Please note that this criterion restricts the overall global volume change of the fluid to 0.1%,

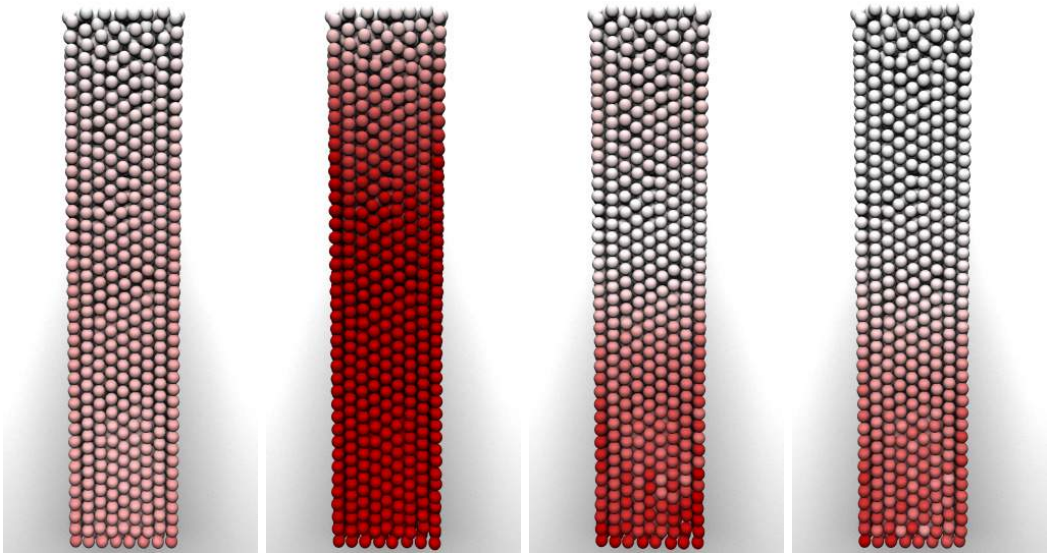

Figure 3.3: Subsequent frames of a resting fluid pillar of 5 meters height, simulated at low resolution. The convergence criterion is set to a maximum density error of 1%. This leads to oscillations in the pressure field and, thus, the free surface. Pressure values are color coded.

but tolerates locally delimited maximum density errors. The average density error η, also referred to as density deviation, compression or volume compression, is consistently used as convergence criterion in all experiments with all solvers.

3.6.2 Performance Comparisons

We present comparisons with three existing solvers. First, we illustrate the performance of IISPH by comparing it to PCISPH, the current state-of-the-art SPH solver in computer graphics. Second, we illustrate the benefits of our PPE formulation by comparing to a state-of-the-art discretization of the PPE as used in ISPH [SL03]. Third, we briefly compare to constraint fluids [BLS12] due to similarities in the concept.

Comparison to PCISPH

We compare IISPH to PCISPH in two scenarios with different particle radii r and different average density errors η to illustrate the effect of these parameters on the time step and the convergence rate of the solvers. Figure 3.4 shows a breaking dam with 100K particles, $r = 0.025$m, and $\eta = 0.01\%$. Figure 3.5 shows a blender with 90K particles, $r = 0.05$m, and $\eta = 0.1\%$. The performance measurements are summarized in Table 3.1 and Table 3.2.

Pressure solve. Compared to PCISPH, IISPH computes the pressure field up to 6.2 times faster for the breaking dam and up to 5.2 times faster in the blender scene. These speed-ups are a combination of an improved convergence and an improved efficiency per iteration. Regarding the convergence, IISPH requires up to 3.6 and 3.1 times less iterations compared to PCISPH. Combined with the improved efficiency per iteration, a speed-up of up to 6.2 and 5.2 is obtained. Further, the speed-ups grow for larger time steps. This indicates that the convergence of IISPH scales better compared to PCISPH for growing time steps.

	PCISPH			IISPH			PCISPH / IISPH		
		comp. time [s]			comp. time [s]		ratio		
Δt [s]	∅ iter.	pressure	total	∅ iter.	pressure	total	iter.	pressure	total
0.0005	4.3	540	1195	2.2	148	978	2.0	3.6	1.2
0.00067	7.2	647	**1145**	2.9	149	753	2.5	4.3	1.5
0.001	14.9	856	1187	4.9	164	576	3.0	5.2	2.1
0.0025	66.5	1495	1540	18.4	242	410	3.6	**6.2**	3.8
0.004	-	-	-	33.5	273	**379**	-	-	-
0.005	-	-	-	45.8	297	383	-	-	-

Table 3.1: Comparison of IISPH with PCISPH using different time steps for a breaking dam with 100K particles (Figure 3.4). Timings are given for the whole simulation (300 frames). The largest ratio in the pressure computation time and the lowest total computation times are marked bold. The maximum volume compression was set to 0.01%. The particles radius was 0.025m.

	PCISPH			IISPH			PCISPH / IISPH		
		comp. time [s]			comp. time [s]		ratio		
Δt [s]	∅ iter.	pressure	total	∅ iter.	pressure	total	iter.	pressure	total
0.0025	4.2	416	978	2.0	106	709	2.1	3.9	1.4
0.005	15.4	688	**964**	4.9	132	435	3.1	**5.2**	2.2
0.01	-	-	-	13.2	182	**338**	-	-	-
0.02	–	-	-	78.8	510	588	-	-	-

Table 3.2: Comparison of IISPH with PCISPH using different time steps for the blender scene with 90K particles (Figure 3.5). Timings are given for the whole simulation (1000 frames). The largest ratio in the pressure computation time and the lowest total computation times are marked bold. The maximum volume compression was set to 0.1%. The particles radius was 0.05m.

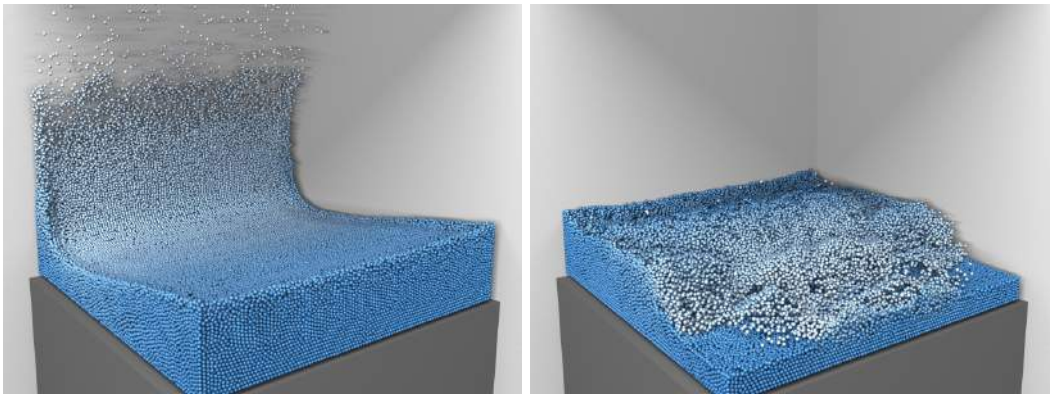

Figure 3.4: Breaking dam with 100K particles.

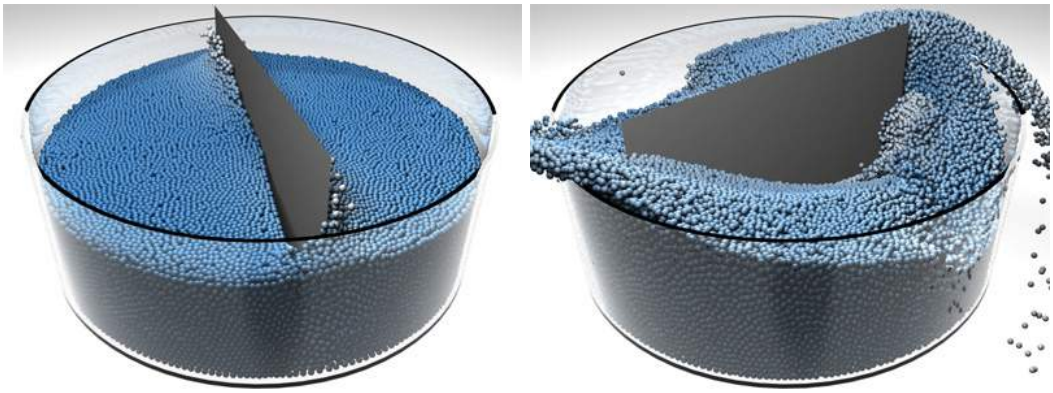

Figure 3.5: Blender scenario with 90K particles.

SPH computation time per simulated frame. IISPH computes the pressure field up to six times faster than PCISPH. However, the overall speed-up in the computation time per simulated frame also depends on other SPH components, e.g. the neighborhood query. The neighborhood query adds roughly constant costs per simulation step. However, the total costs per simulated frame for finding the neighbors decrease with larger time steps as less simulation steps are performed per simulated frame. Interestingly, increasing costs for the pressure solve and decreasing costs for the neighborhood search result in the fact, that IISPH and PCISPH do not necessarily achieve their best overall performance for the largest possible time step. For the breaking dam, IISPH and PCISPH handle time steps of up to 0.005s and 0.0025s, respectively. The factor of two with respect to the maximum time step is, however, practically irrelevant for the overall computation time, as both solvers achieve their best performance for time steps of 0.004s and 0.00067s, corresponding to a factor of six with respect to the optimal time step and a factor of three with respect to the overall performance gain. The situation in the blender scenario is analogous.

Time step. As discussed in [SP09], PCISPH scales pressure values by a global stiffness value δ which is time-step dependent. So, dependent on the time step, δ might be too large and cause overshooting, or it might be too small and negatively influence the convergence rate. The performance and stability of PCISPH are, thus, heavily influenced by the time step. For IISPH, the time step has less effect on the performance. The influence of the time step on the number of iterations is illustrated in Figure 3.6.

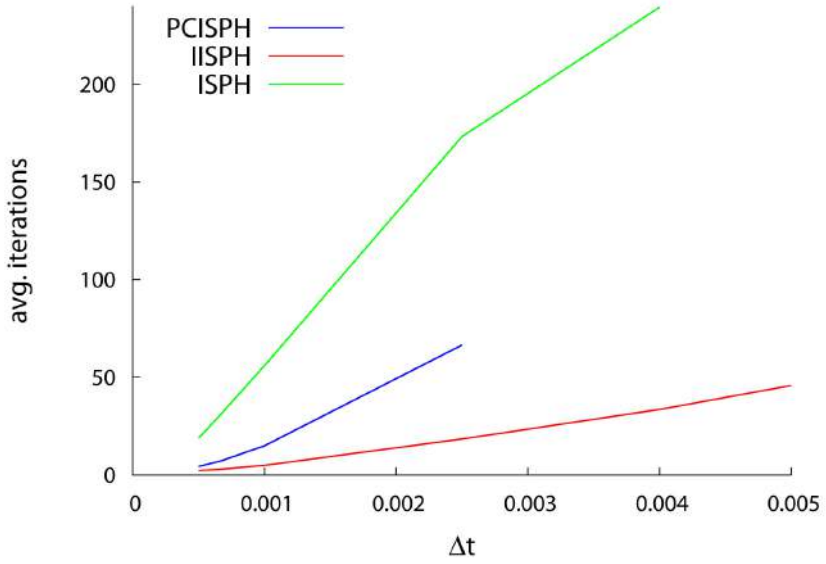

Figure 3.6: Average number of iterations of PCISPH, IISPH and ISPH for different time steps in the breaking dam scenario.

For the performance comparisons (Table 3.1 and Table 3.2), we kept the time step fix for each simulation run. However, in practice, adaptive time-stepping schemes are desirable. For PCISPH, the coupling of the time step size with the convergence rate and the stability is an issue that requires sophisticated techniques to realize adaptive time steps, e.g. [IAGT10]. In contrast, changing time steps do not negatively affect the robustness of IISPH. The time step can just be set according to the CFL condition, e.g. $\Delta t = \min\left(0.4h/\left|\mathbf{v}_i^{adv}\right|\right)$, without rolling back the simulation as required in PCISPH [IAGT10, RWT11]. In practice, this has a positive effect on the overall performance gain of IISPH compared to PCISPH.

Comparison to ISPH

IISPH differs from ISPH presented in [SL03] in the discretization of the Laplacian and the source term. Both aspects improve the performance of IISPH. While the proposed discretization of the Laplacian improves the convergence rate compared to [SL03], the IISPH source term allows for larger time steps compared to the source term of [SL03]. We first summarize the ISPH formulation of [SL03], followed by an analysis of the benefits of the proposed IISPH formulation.

Discretization of the Laplacian. IISPH takes contributions of second-ring neighbors into account which improves the convergence rate compared to ISPH. In order to verify this, we compared the left-hand side of (3.18) with the approximate discretization of the Laplacian in ISPH (3.12). In this comparison, we only vary the discretization of the Laplacian. The source term is taken from (3.18) in both cases. Figure 3.7 illustrates the convergence of both implementations for one simulation step of the breaking dam scenario with equivalent initial particle configurations. IISPH achieves an estimated compression of 0.01% after 23 iterations, while ISPH takes 231 iterations. As we only use different discretizations of the Laplacian in both settings, the experiment indicates that the specific form of the Laplacian in IISPH significantly improves the convergence rate compared to ISPH. While Figure 3.7

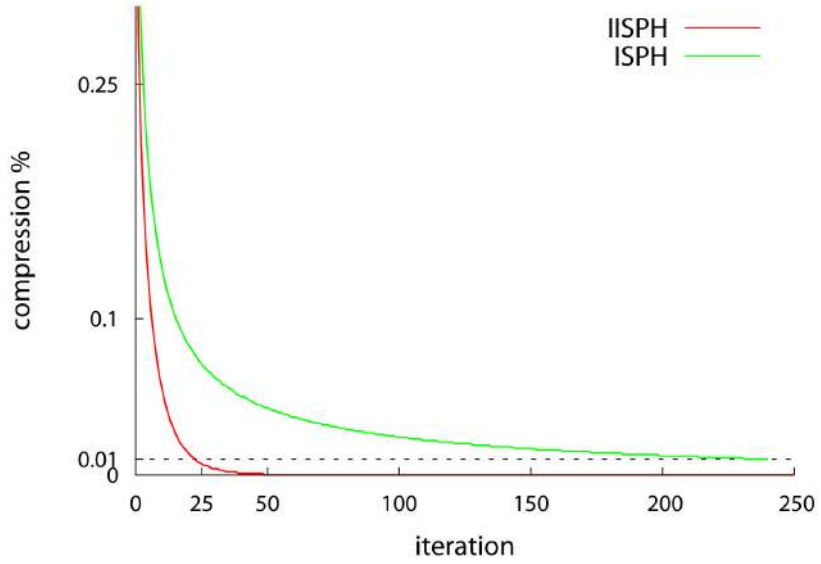

Figure 3.7: Convergence of IISPH and ISPH for one simulation step of the breaking dam scenario with $\Delta t = 0.0025s$.

compares the convergence for one simulation step, Figure 3.6 compares the convergence rate for varying time steps using the same setup. The experiments show in all cases that the convergence rate of the IISPH discretization is superior to the ISPH discretization.

Furthermore, the approximation of the Laplacian (3.12) employed in ISPH does not accurately consider the pressure force, but uses different approximations for the pressure gradient and the Laplacian. Therefore, the predicted level of compression when solving the PPE might significantly differ from the real compression at time $t + \Delta t$. For the breaking dam scene, the obtained density error of ISPH is 0.29% for a time step of 0.004s. In contrast, the error introduced by the semi-implicit formulation employed in IISPH is much smaller. IISPH could even obtain a compression of 0.011% for the largest time step of 0.005s.

Source term. IISPH employs velocity projection in the source term. In order to illustrate the positive effect of this choice on the time step, we have compared velocity projection (3.17) with position projection (3.2) as proposed in [SL03]. For the comparison, we use the proposed IISPH formulation in (3.18) for the Laplacian. For the breaking dam scenario, the setting with position projection could only handle time steps up to 0.001s, whereas four times larger time steps can be handled with velocity projection.

Performance. Finally, we compare the performance of ISPH [SL03] to IISPH on the breaking dam scenario. The timings of ISPH are given in Table 3.3, while the timings of IISPH are given in Table 3.1. ISPH performs best at a time step of 0.001s for the breaking dam scenario with an overall computation time of 1535s where 1123s is represented by the pressure computation. For the same time step, IISPH computes the pressure field in 164s which is 6.8 times faster than ISPH. IISPH performs optimal at a time step of 0.004s with 379s. This is an overall speed up of 4 compared to the ISPH formulation presented in [SL03].

		total comp. time [s]		
Δt [s]	avg. iter.	pressure	overall	real compression
0.0005	18.8	759	1588	0.010%
0.00067	31.1	959	1563	0.010%
0.001	55.9	1123	1535	0.018%

Table 3.3: Performance of ISPH [SL03] for the breaking dam scenario (Figure 3.4). The right column gives the obtained compression. The tolerated error was set to 0.01%.

Comparison to Constraint Fluids

Another related approach for incompressible SPH is presented in [ESE07], [BLS12], and [MM13]. Similar to the proposed method, incompressibility is enforced by imposing density constraints. In contrast to ISPH and IISPH, the incompressibility condition is not derived from the continuity equation, but from constraint dynamics. Density constraints are enforced through the use of Lagrange multipliers and not by employing standard SPH formulations for pressure and pressure force.

In [BLS12], a setup of a fluid pillar with 10K particles of radius $r = 0.01$m is presented. For a time step of 0.0083s and a fixed number of 15 pressure iterations, a volume compression of 17% has been reported. We simulated this test case with IISPH using the parameters given in [BLS12]. For the same number of pressure iterations per frame, IISPH enforced a volume compression of 1.2% which is 14 times less compression than stated for the constraint-based method. This indicates an improved convergence of the IISPH pressure solver compared to [BLS12].

Comparison to hybrid methods

A comparison with [RWT11] is not considered in this thesis. On one hand, the performance of [RWT11] is not only influenced by the number of particles, but also by the grid complexity which governs the size of the linear system. On the other hand, the performance of the IISPH solver heavily depends on the user-defined density error per particle, which is not discussed in [RWT11]. The performance dependency on the tolerated compression per particle is also the reason, why a comparison to FLIP, e.g. [ZB05], is not appropriate, where the density deviation cannot be computed per particle.

3.6.3 Large-scale Scenarios

Poor performance scaling has been reported for all previous global SPH methods, e.g., for ISPH in [CR99] and for constraint fluids in [ESE07]. The most complex scenarios presented in the most recent constraint approaches [BLS12] and [MM13] comprise 250K and 128K particles, respectively. In contrast, the performance of IISPH scales well with the size of the simulation domain as the convergence and the costs per particle are invariant to the number of particles. This is verified on a large-scale scenario where a city with an area of $5 \cdot 10^6 \text{m}^2$ is flooded at two different resolutions (Figure 3.8). In the low resolution, we simulated up to 6 million particles with a radius of 1 meter. The high resolution contains up to 40 million particles with radius 0.5 meter. The factor of 6.7 in the number of particles instead of 8 stems from the shallow parts of the simulation. The IISPH solver took 5.7s to

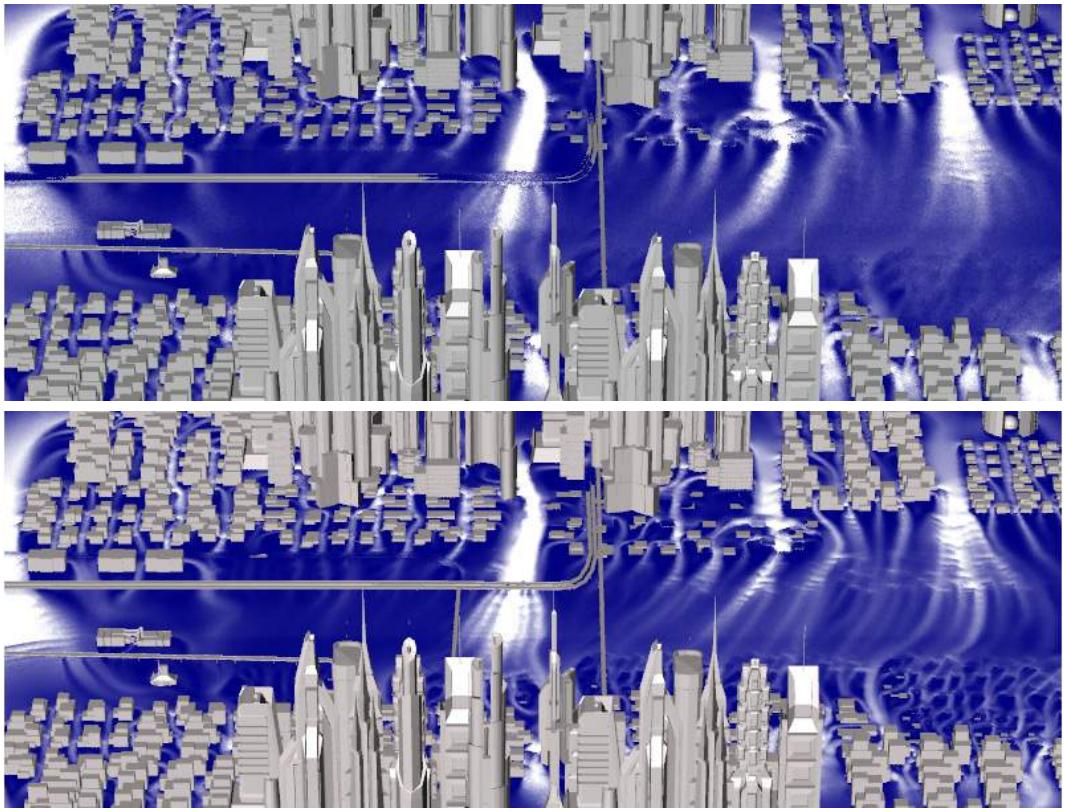

Figure 3.8: A city with an area of $5 \cdot 10^6 \mathrm{m}^2$ is flooded at low resolution (top), 6 million particles, and high resolution (bottom), 40 million particles. Velocities are color coded.

update the low-resolution simulation and 38s for the high resolution, see Table 3.4. This is a performance ratio of 6.7 which shows the perfect linear scaling of the proposed method.

The practicability of the approach is further demonstrated by three large-scale scenarios with particle counts ranging from 10 to 30 million including two-way coupling with rigid objects. Performance measurements are given in Table 3.4.

scene	part.	r	avg. Δt	avg. iter.	comp. time / Δt pressure	total
City	6M	1.000m	0.0500s	5.2	2.5s	5.7s
	40M	0.500m	0.0250s	4.1	15.8s	38.2s
Island	10M	0.075m	0.0054s	13.5	6.5s	13.6s
Cargo	30M	0.050m	0.0040s	16.1	24.9s	44.2s
Street	28M	0.025m	0.0025s	17.3	23.1s	41.8s

Table 3.4: Measurements for the large scale scenarios.

Figure 3.9: A scripted island rises out of the water. The simulation contains 10 million fluid particles. One simulation step is computed in 14 seconds with a time step of 0.0054 seconds and a compression of less than 0.1%.

In the Island scene (Figure 3.9), 10 million particles of volume radius $r = 0.075$m were simulated with an average time step of 0.0054s. On average, the simulation update took 13.6s where the pressure solve represents 6.5s. For the larger scenes, Street (Figure 3.10) and Cargo (Figure 3.11), strong turbulences were generated by high-velocity inflows. The Street scene with 28 million particles and $r = 0.025$m was updated in 41.8s where 23.1s were spent on solving the pressure. This is similar to the performance of the Cargo scene, 30 million particles and $r = 0.05$m, where a simulation step is computed in 44.2s with 24.9s represented by the computation of the pressure field.

Figure 3.10: Street flood. This scene contains up to 28 million fluid particles. Using our approach, one simulation step can be computed in 42 seconds with an average time step of 0.0025 seconds and a compression of less than 0.1%.

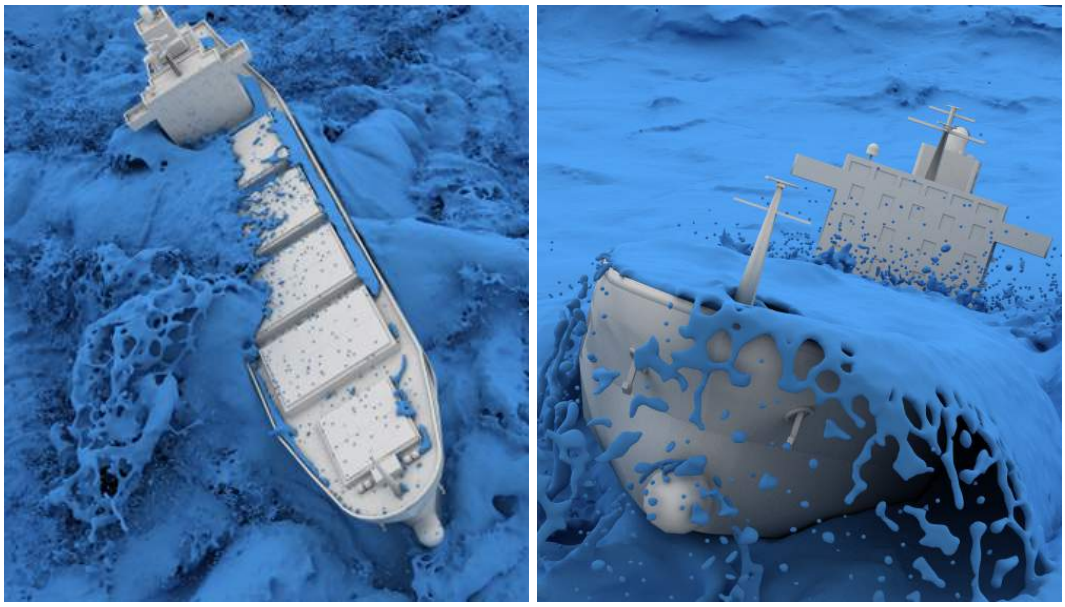

Figure 3.11: A cargo ship in highly agitated water. Up to 30 million particles are processed in 44 seconds per simulation step with a time step of 0.004 seconds and a compression of less than 0.1%.

3.7 Discussion

In this chapter, a novel discretization of the PPE has been presented which can be solved efficiently, handles large time steps and scales well for large-scale scenarios. It has been shown that the accuracy and fast convergence is based on the fact that only few approximations are employed in the derivation of the scheme. Comparisons with ISPH and PCISPH did not only show the superiority of the proposed IISPH method, but also illustrated novel aspects that have not been discussed before. E.g., when embedded in an SPH framework,

the maximum time step that can be handled by an iterative pressure solver generally does not correspond to the most efficient setting.

As discussed, the optimal time step with respect to the performance also depends on the performance of the neighborhood search as the neighborhood has to be updated in each time step. In general, the efficient querying and processing of neighbors is another key for realizing an highly efficient SPH implementation. This topic is addressed in the next chapter.

CHAPTER 4

Neighborhood Search

In SPH, the dynamics of the material is governed by the local influence of neighboring particles. In SPH-based fluid simulations, the set of neighboring particles dynamically changes over time. Therefore, the efficient querying and processing of particle neighbors is crucial for the performance of the simulation. The neighborhood problem for SPH fluids is similar to, e.g., collision detection or intersection tests in ray tracing. However, neighborhood queries in SPH are also characterized by unique properties that motivate the investigation of efficient data structures in this context.

For instance, adjacent cells in the employed spatial data structure have to be accessed efficiently. Further, an efficient data structure should make use of the temporal coherence of a fluid between two subsequent simulation steps, but over larger time periods, it should preserve or restore spatial locality. As the acceleration structure has to be updated in each simulation step, query as well as construction times have to be optimized. Therefore, acceleration structures used in static applications, e.g., kd-trees or perfect hashing, might be too expensive to construct. Querying has to be efficient, as the set of neighboring particles has to be processed multiple times in an SPH algorithm for computing various attributes.

In a typical SPH simulation, a particle is interacting with 30-40 neighbors. Storing the neighbor set for fast reuse is, thus, a natural choice. However, for systems with a low-memory limit, this quickly limits the maximum complexity of the scene. In fact, some SPH implementations on the GPU [HKK07b, ZSP08, GSSP10] do not store interacting particles, but recompute neighbor sets when needed. Consequently, the performance of these systems scales with the number of neighborhood queries in each simulation step. Therefore, either the gas equation [MCG03] or the Tait equation [Mon94, BT07] are employed, as these equations are fast to compute and the influencing particle sets are processed only twice per simulation step. However, as discussed in Chapter 3, the time step limit for incompressible simulations results in a low overall performance for EOS-based SPH implementations.

More efficient SPH simulations such as IISPH [ICS+13] and PCISPH [SP09], tolerate significantly larger time steps, but each update is more expensive to compute. Due to the iterative scheme, the neighbor set is processed multiple times per simulation step which might limit the performance if the proximity information can not be queried efficiently.

This chapter presents an efficient system for computing SPH fluid simulations on multi-core CPUs. Since the query and processing of particle pairs is crucial for the performance, we focus on parallel, spatial acceleration structures. We propose compact hashing and Z-index sort as optimizations of the commonly used spatial hashing [THM+03] and index sort [Gre08] schemes. The different techniques are described in Section 4.2.

A low memory transfer is evident for a good performance of parallel systems. We employ

techniques that lower the latency when accessing particles and their interacting neighbors. Temporal coherence is exploited in order to reduce grid construction times for both acceleration structures. We show that the performance of the proposed compact hashing is competitive with Z-index sort, while compact hashing is more appropriate for large scale fluid simulations in arbitrary domains.

In Section 4.3, the performance of compact hashing and Z-index sort is thoroughly analyzed in the context of SPH. Both proposed schemes are compared with three existing variants of uniform grids, i.e., basic uniform grid, spatial hashing, and index sort.

4.1 Related Work

Since the introduction of SPH-based fluids to the graphics community [MCG03], research in this field is focusing on performance aspects of SPH.

There exist various approaches that address the implementation of SPH on highly parallel architectures, especially on the GPU. While SPH computations are easy to parallelize, the implementation of an efficient parallelized neighborhood search is not straightforward. In [AIY+04], e.g., the GPU is only used for computing the SPH update but not for the neighborhood query. Later, [HKK07b, ZSP08] presented SPH implementations that run entirely on the GPU. These methods map a three-dimensional uniform grid onto a two-dimensional texture. Particle indices are stored in RGBA channels. Thus, only four particles can be mapped into each grid cell. Due to this issue, a smaller than optimal cell size has to be chosen in order to avoid simulation artifacts.

Generally, even though a uniform grid is fast to construct and the costs of accessing an item are in $O(1)$, it suffers from a low cache-hit rate in the case of SPH fluid simulations. This is due to the fact that the fluid fills the simulation domain in a non-uniform way. Only a small part of the domain is filled, while the fluid also tends to cluster. Consequently, the grid is only sparsely filled and, hence, a significant amount of memory is unnecessarily allocated for empty cells. As stated in [HKK07b], this decreases the performance due to a higher memory transfer. In [HKK07a], an adaptive uniform grid is presented, where the memory consumption scales with the bounding box of the fluid volume.

Index sort is another approach to reduce the memory consumption and transfer of the uniform grid. Index sort first sorts the particles with respect to their cell indices c. Then, indices of the sorted array are stored in each cell. Each cell just stores one reference to the first particle with corresponding cell index. This idea has been described by Purcell et al. [PDC+03] for a fast photon mapping algorithm on the GPU. In [Ov08], a similar idea is applied to a non-parallel SPH simulation. The paper shows that index sort outperforms the spatial hashing scheme for simulations with a low number of 1300 particles. Index sort is used in NVIDIA's CUDA-based SPH fluid solver [Gre08] and is also employed for fast parallel ray tracing of dynamic scenes [LD08, KS09]. In this work, we discuss important issues of the index-sort scheme in the context of a parallel SPH implementation on multi-core CPUs. Thereby, we propose a new variant named *Z-index sort* which includes SPH-specific enhancements.

In contrast to the uniform grid, the spatial hashing method can represent infinite domains. This method has been introduced for deformable solids [THM+03] and rigid bodies [GBF03]. For particle-based granular materials, spatial hashing is applied by [BYM05]. We propose a memory-efficient data structure for the spatial hashing method called *compact hashing* that allows for larger tables and faster queries.

In general, a hash function does not maintain spatial locality of data, which increases the memory transfer. In order to enforce spatial locality, we employ a space-filling Z-curve. A similar strategy is used by Warren and Salmon [WS95]. In this approach, a hashed octree is presented for parallel particle simulations. For the cosmological simulation code GADGET-2, particles are ordered in memory according to a Peano-Hilbert curve [Spr05]. The Peano-Hilbert curve preserves the locality of the data very well, but is relatively expensive to evaluate. Instead, we use the Z-curve which can be efficiently computed by bit-interleaving [PF01].

In addition to the uniform grid, spatial hashing and index sort, various other acceleration structures have been presented. One of the most popular techniques is the Verlet-list method [Ver67, Hie07]. In this method, a list of potential neighbors is stored for each particle. Potential neighbors have a distance which is lower or equal to a predefined range s, where s is significantly larger than the influence radius r. Thereby, the list of potential neighbors needs to be updated only if a particle has moved more than $s - r$. However, the memory transfer of this method scales with the ratio s/r, as all potential particle pairs are processed in each neighborhood query.

In [KW06], particles are sorted according to staggered grids, one for each dimension. Instead of querying spatially close cells in all dimensions at once, dimensions are processed one after another. As stated in [KW06], this method works well for a low number of particles. For higher resolutions, the performance is lower compared to, e.g., the octree used in [VCC98]. However, as reported in [PDC+03] and [HKK07b], hierarchical subdivision-schemes are not a good choice for fluids with uniform influence radius since the costs of accessing an item are in $O(\log n)$, while for uniform grids they are in $O(1)$. This implies a higher memory transfer which specifically limits the performance of hierarchical data structures in parallel implementations. This particularly holds for kd-trees which have been employed by Adams et al. [APKG07] for adaptively sampled SPH. The idea of this method is to reduce the number of particles inside a volume in order to improve the performance for densely sampled fluids. However, in order to efficiently find interacting particles with different influence radii, the kd-tree is rebuilt in every time step. According to the presented timings, the neighborhood search marks the performance bottleneck for this method.

The above-mentioned GADGET-2 simulation code is designed for massively parallel architectures with distributed memory. MPI (Message Passing Interface) is used for the parallelization. A parallel library for distributed memory systems is also presented by Sbalzarini et al. [SWB+06]. Using the library, continuum systems can be simulated with different particle-based methods. [FE08] use orthogonal recursive bisection for decomposing the simulation domain in order to achieve load-balanced parallel simulations of particle based fluids. This approach is designed for cluster systems. While these approaches focus on distributed memory systems, our approach addresses shared memory systems, where parallelization can be efficiently realized using OpenMP [Ope05].

4.2 Data Structures

The neighborhood search can be accelerated using spatial subdivision schemes. As reported in [HKK07b], the construction cost of a hierarchical grid is $O(n \log n)$ and the cost of accessing a leaf node is $O(\log n)$. In contrast, the construction cost of a uniform grid is $O(n)$, while any item can be accessed in $O(1)$. Therefore, uniform grid approaches are most efficient for SPH simulations with uniform support radius h.

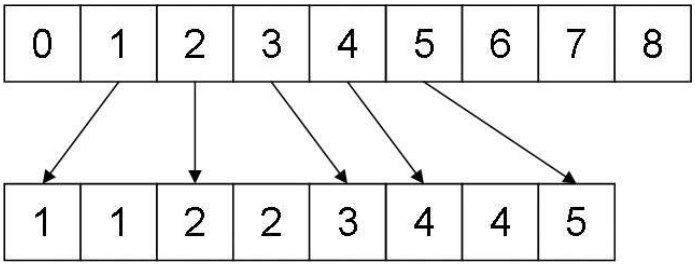

Figure 4.1: Index sort data structure. The top row illustrates the uniform grid where the numbers refer to the corresponding cell indices. Each non-empty cell points to the first particle in the sorted particle array with corresponding cell index (bottom row).

4.2.1 Basic Uniform Grid

In the basic uniform grid approach, each particle i with position $\mathbf{x} = (x, y, z)$ is inserted into **one** spatial cell with coordinates (k, l, m). In order to determine the neighborhood of i, only particles that are in the same spatial cell or in one of the neighboring spatial cells within distance h need to be queried.

Obviously, the cell size d has an impact on the number of potential neighbors. The smaller the cell size, the smaller the number of potential pairs. However, with cell sizes smaller than h, the number of cells to query gets bigger. This might slow down the neighborhood query due to a larger number of memory lookups. In the following, we assume a cell size of h. In this case, 27 cells have to be queried for each particle. We discuss the performance of different cell sizes in Section 4.3.

4.2.2 Index Sort

Parallel construction. The parallel construction of the uniform grid is not straightforward since the insertion of particles into the grid might cause race conditions, i.e., two or more threads try to write to the same memory address concurrently.

As suggested in [PDC+03, KS09], these memory conflicts can be avoided by using sorted lists. The index sort method first sorts the particles in memory according to their cell index c. The cell index c of a position $\mathbf{x} = (x, y, z)$ is computed as

$$c = k + l \cdot K + m \cdot K \cdot L \tag{4.1}$$
$$\mathbf{c} = (k, l, m) = \left(\left\lfloor \frac{x - x_{min}}{d} \right\rfloor, \left\lfloor \frac{y - y_{min}}{d} \right\rfloor, \left\lfloor \frac{z - z_{min}}{d} \right\rfloor \right)$$

with d denoting the cell size. K and L may either denote the number of cells in x and y direction of the fluid's AABB or of the whole simulation domain.

In contrast to non-sorted uniform grids, a grid cell does no longer store references to all the particles in this cell. In fact, a cell with index c does only store one reference to the first particle in the sorted array with cell index c. We use a parallel reduction to insert the references into the grid [KS09]. Thereby, each bucket entry $B[k]$ in the sorted particle array B is tested against $B[k-1]$. Let c_k denote the cell index of the particle stored at $B[k]$. If c_k is different from c_{k-1}, a reference to $B[k]$ is stored in the spatial grid cell $G[c_k]$. This procedure can be performed in parallel since race conditions do not occur. The final data structure is illustrated in Figure 4.1.

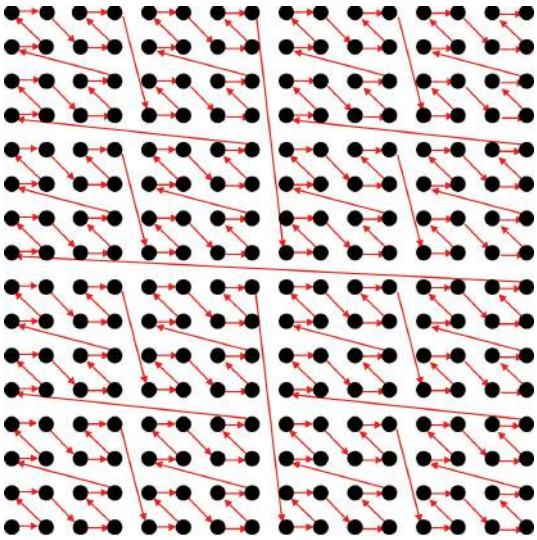

Figure 4.2: The self-similar structure of the Z-curve in 2D is illustrated for a regular grid.

Index sort avoids expensive memory allocations while the memory consumption is constant. However, the whole spatial grid needs to be represented in order to find neighboring cells.

Parallel query. The neighborhood query for parallel architectures is straightforward. The sorted particle array is processed in parallel. For each particle i, all particles in the 27 neighboring cells are tested for interaction. Due to the sorting, particles that are in the same spatial cell are also close in memory. This improves the memory coherence of the query. However, it depends on the indexing scheme if particles in neighboring cells are also close in memory. In the following section, we discuss important aspects when applying the index sort method on parallel CPU architectures.

4.2.3 Z-index Sort

Indexing scheme. Current shared-memory parallel computers are built with a hierarchical memory system, where a very fast, but small cache memory supplies the processor with data and instructions. Each time new data is loaded from main memory, it is copied into the cache, while blocks of consecutive memory locations are transferred at a time. A new block may dispose other blocks of data, if the cache is full. Operations can be performed with almost no latency if the required values are available in the cache. Otherwise, there will be a delay. Thus, the performance of an algorithm is improved by reducing the amount of data transferred from main memory to the cache. The transfer rate decreases for higher cache-hit rates, i.e., the percentage of accesses where the requested data is already present in the cache. Consequently, for parallel algorithms we have to enforce that threads do share as much data as possible without generating race conditions.

Even though it is not possible to directly control the cache, the program can be structured such that the memory transfer is reduced. The indexing scheme defined in (4.1) is not spatially compact since it orders the cells according to their z position first. Thus, particles that are close in space are not necessarily close in memory. In order to further reduce the memory transfer, we suggest to employ a space-filling Z-curve for computing the cell indices.

Rather than sorting an n-dimensional space one dimension after another, a Z-curve orders the

space by n-dimensional blocks of 2^n cells. This ordering preserves spatial locality very well due to the self-containing (recursive) block structure (see Figure 4.2). Consequently, it leads to a high cache-hit rate while the indices can be computed fast by bit-interleaving [PF01]. In Section 4.3, we show that the Z-curve improves the cache-hit rate and, therefore, improves the performance for the query and processing of particle neighbors.

Sorting. The particles carry several physical attributes, e.g., velocity, position and pressure. When sorting the particles, these values have to be copied several times. Thus, the memory transfer might slow down the sorting significantly. In order to avoid sorting the particles array in every simulation step, we suggest to use a secondary data structure which stores a *key-value* pair, called handle. Each handle stores a reference to a particle (value) and its corresponding cell index (key). Due to the minimal memory usage of this structure, sorting the handles in each time step is much more efficient than sorting the particle array.

In order to yield high cache-hit rates, the particle array itself should still be reordered. However, we experienced that it is sufficient to reorder the particle array every 100th simulation step since the fluid simulation evolves slowly and coherently. The temporal coherence of the simulation data can be exploited further. According to the Courant-Friedrich-Levy (CFL) condition, a particle must not move more than half of its influence radius h in one time step. Thus, the average number of particles for which the cell index changes in a consecutive simulation step is low, i.e., 2%. Therefore, we propose to use *insertion sort* for reordering the handles. Insertion sort is very fast for almost sorted arrays. Usually, parallel *radix sort* is used for sorting [Gre08, Ov08]. As we show in Section 4.3, the insertion sort outperforms our parallel radix sort implementation on CPUs even for large data sets.

Generally, index sort variants are considered to be the fastest spatial acceleration methods. However, according to [Gre08] there are two limitations. First, the memory consumption scales with the simulation domain. Second, for sorting, the whole particle array has to be reprocessed after computing the cell indices. In the following sections, we discuss an acceleration structure which is able to represent infinite domains.

4.2.4 Spatial Hashing

In order to represent infinite domains with low memory consumptions, we employ the *spatial hashing* procedure introduced in [THM+03]. In this scheme, the effectively infinite domain is mapped to a finite list. The hash function that maps a position $\mathbf{x} = (x, y, z)$ to a hash table of size m has the following form:

$$c = \left[\left(\left\lfloor \frac{x}{d} \right\rfloor \cdot p_1 \right) \text{xor} \left(\left\lfloor \frac{y}{d} \right\rfloor \cdot p_2 \right) \text{xor} \left(\left\lfloor \frac{z}{d} \right\rfloor \cdot p_3 \right) \right] \% m, \tag{4.2}$$

with p_1, p_2, p_3 being large prime numbers that are chosen similar to [THM+03] as 73856093, 19349663 and 83492791, respectively.

Unfortunately, different spatial cells can be mapped to the same hash cell (hash collision), slowing down the neighborhood query. The number of hash collisions can be reduced by increasing the size of the hash table. According to [THM+03], the hash table should be significantly larger than the number of primitives in order to minimize the risk of hash collisions. Our experiments indicate that for SPH, a hash table size of two times the number of particles is appropriate.

In order to avoid frequent memory allocations, [THM+03] suggests to reserve memory for a certain number k of entries in all hash cells on initialization. Thereby, a table with m hash cells consumes $O(m \cdot k)$ memory. However, on average, the number of non-empty cells is only

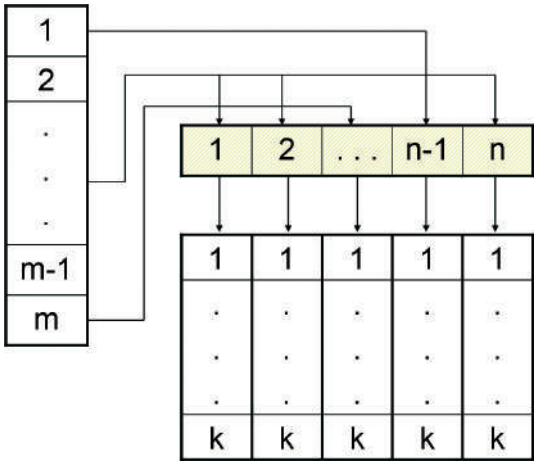

Figure 4.3: Compact hashing. A handle array is allocated for the hash table of size m. The handles point to a compact list which stores the small number of n used cells. For a used cell, memory for k entries is reserved. The memory consumption is thereby reduced to $O(n \cdot k + m)$. Note that the neighborhood query only traverses the n used cells.

12% of the total number of particles. For SPH fluids, the hash table is generally sparsely filled and a significant amount of memory is unnecessarily pre-allocated. Furthermore, for most of the hash cells, adjacent cells might be empty which reduces the cache-hit rate for the insertion and query. In the following section, we present solutions to these problems.

4.2.5 Compact Hashing

In order to reduce the memory consumption, we propose to use a secondary data structure which stores a compact list of non-empty (used) cells. The hash cells do only store a handle to their corresponding used cell. Memory for a used cell is allocated if it contains particles and deallocated if the cell gets empty. Thus, this data structure consumes constant memory for the hash table storing the handles and additional memory for the list of used cells. In contrast to the basic uniform grid, the memory consumption scales with the number of particles and not with the simulation domain. The data structure is illustrated in Figure 4.3.

The compact list of used cells already lowers the expected memory transfer. However, the hash function is not designed to maintain spatial locality, but rather abolishes it. Thus, compared to index sort, the required memory transfer for the query is still comparatively large. This again results in much larger query times, even if there are no hash collisions. In the following, we propose techniques which significantly improve the performance of the construction and query for the spatial hashing method.

Parallel construction. Similar to non-sorted uniform grid, the particles cannot be inserted into the used cells in parallel. On the other hand, temporal coherence can be exploited more efficiently. Therefore, we do not reconstruct the compact list in each time step, but only account for the changes in each cell.

In a first loop, the spatial cell coordinates $\mathbf{c}_i = (k, l, m)$ of each particle i are computed. Only if $\mathbf{c}_i$ has changed, the new cell index of i is computed with (4.2) and the particle is added to a list of *moving particles*. In a second step, the moving particles are removed from their old cell and inserted into the new one. Note that the second step has to be performed

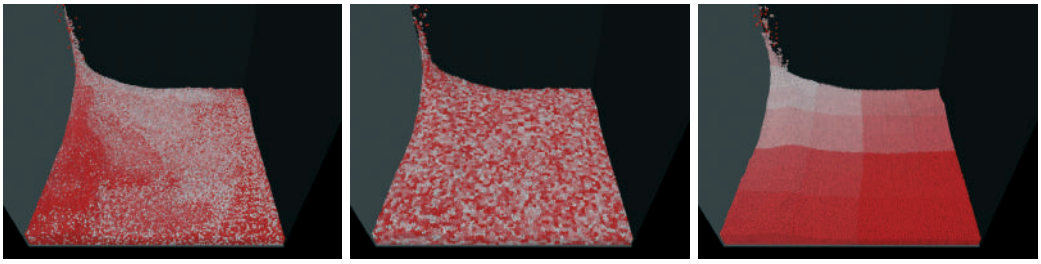

Figure 4.4: Particles are colored according to their location in memory. Spatial locality is not maintained if particles are not reordered (left). Since the hash function abolishes spatial locality, reordering according to the cell index (middle) is a bad choice to reduce the memory transfer. Spatial compactness is enforced using a Z-curve (right).

serially, but with very low amortized costs since the number of moving particles is generally small, i.e., around 2% on average.

Parallel query. The neighborhood query is straightforward. The compact list of used cells can be processed in parallel without any race conditions. Nevertheless, the query is expected to be comparatively slow for the spatial hashing. This is due to hash collisions and an increased memory transfer as consecutive used cells are close in memory, but not close in space. We now give solutions to overcome these limitations.

Hash collisions. For a used cell without hash collisions, all particles are in the same spatial cell and, hence, the potential neighbors are the same. However, if there is a hash collision in a used cell, the hash indices of the neighboring spatial cells have to be computed for each particle. This results in a significant computational overhead, particularly since we do not know if there is a hash collision in a cell or not. In order to keep the overhead low, we suggest to store a hash-collision flag in each used cell. Therefore, the hash indices have to be computed only once for cells without hash collisions.

The memory consumption of the proposed data structure scales with the number of particles and not with the hash (handle) table size. Therefore, the hash table can always be set to a size which enforces a very low number of hash collisions. For SPH, this number is usually below 2%. Storing a collision flag significantly speeds up the query since the performance is almost not influenced by the low number of hash collisions.

Spatial locality. Hash functions are designed to map data from a large domain to a small domain. Thereby, the data is scattered and spatial locality is usually not preserved. The list of used cells is, hence, not ordered according to space which means that consecutive cells are not spatially close (see Figure 4.4). Thus, an increased memory transfer is expected for the query since already cached data might not be reused.

In order to reduce the memory transfer, we again sort the particles according to a Z-curve every 100th simulation step. Sorting is performed similar as described in Section 4.2.3. Instead of sorting the particle array with all attributes, we sort a handle structure. Each handle consists of cell index and reference to a particle. Then, all other attributes of the particle are sorted accordingly.

Note that when sorting the particle array, the pointers in the used cell become invalid. Therefore, the compact used-cell list has to be rebuilt. Used cells are created each time a particle is added to an empty hash cell. Thus, if we simply insert the sorted particles serially into the compact hash table, the order of the used cells is consistent with the Z-curve order

method	construction	query	total
basic uniform grid	25.7 (27.5)	38.1 (105.6)	**63.8** (133.1)
index sort [Gre08]	35.8 (38.2)	29.1 (29.9)	**64.9** (77.3)
Z-index sort	16.5 (20.5)	26.6 (29.7)	**43.1** (50.2)
spatial hashing	41.9 (44.1)	86.0 (89.9)	**127.9** (134.0)
compact hashing	8.2 (9.4)	32.1 (55.2)	**40.3** (64.6)

Table 4.1: Performance analysis of different spatial acceleration methods with and without (in brackets) reordering of particles. Timings are given in milliseconds for CBD 130K and include storing of pairs.

of the particles. We employ a similar parallel reduction scheme as in Section 4.2.2, in order to parallelize this step. The computational overhead of reconstructing the compact list of used cells is negligible since reordering every 100th steps is sufficient.

In this section, we described a spatial hashing scheme optimized for SPH simulations on parallel architectures. By allocating memory only for non-empty hash cells, the overall memory consumption is reduced. Furthermore, for the neighborhood query, only the small percentage of used cells is traversed which significantly improves the performance. Finally, for shared memory systems, the memory transfer is minimized by reordering particles and the compact list according to a space-filling curve.

4.3 Results

In this section, we evaluate the performance of the investigated data structures. We start with a detailed performance analysis of the five uniform grid variants. By comparing the construction and query times, we show that the proposed Z-index sort and the compact hashing are most efficient. Next, we discuss the performance influence of the cell size and the scaling of our system for a varying number of threads. Finally, we show that mapping spatial locality onto memory improves the efficiency of the solver. This is demonstrated using the example of SESPH and PCISPH which are described in Section 2.4 and Section 3.2.1, respectively.

Our test system has the following specifications:

- CPU: 4x 64-Bit Intel Six Core 7460@2.66GHz, 16 MB shared L3 cache, 64MB Snoop Filter that manages cached data to reduce data traffic
- Memory: 128GB RAM, Bus Speed 1066MHz

4.3.1 Performance Analysis

We use **one** example scene for analyzing the performance of the presented spatial acceleration methods. For this scene, we have chosen a corner breaking dam with 130K particles (CBD 130K) since it is a typical test scene in the field of computational fluid dynamics. We averaged the performance over 20K time steps (see Table 4.1).

Basic uniform grid. The construction of the basic uniform grid is not parallelized due to race conditions. The query step loops over all particles in parallel and computes the neighbors. Thereby, we do not have to traverse the whole, sparsely filled grid. However, if

method	compact hashing	PCISPH (SESPH)	ups
no reorder	64.6	171.8 (38.4)	4.2 (9.6)
reorder	40.3	78.6 (17.6)	8.3 (16.7)

Table 4.2: Performance improvement when particles are reordered every 100 steps. *ups* denotes simulation updates (neighbor search + fluid update) per lab second. Timings are in milliseconds for CBD 130K.

particles are not reordered, the memory transfer is high which has a significant impact on the performance.

Index sort. In comparison to the basic uniform grid, the query of the index sort is much faster since the query processes the particles in the order of their cell indices. Consequently, the cache-hit rate is high, which improves the performance significantly. However, the construction time is comparatively large due to a slow sorting time. Our parallel radix sort algorithm sorts the 130K key-value pairs (handles) in 25ms which is slow compared to the much faster multi-core sort implementation described in [CNL+08]. Note that for this method, particles are reordered according to their cell index which is computed with (4.1).

Z-index sort. For Z-index sort, the handles are sorted using insertion sort instead of radix sort. Due to the small changes from one time step to the next, only a small number of particles moves to another spatial cell. Thus, the handles are only slightly disordered from one time step to the next. Since insertion sort performs very well on almost sorted lists, the sorting time is significantly reduced to 6ms. Furthermore, in contrast to index sort, the cell indices are computed on a Z-curve and the particles are reordered accordingly. By mapping spatial locality onto memory, the cache-hit rate is increased and, hence, the construction and query time for Z-index sort is further reduced.

Spatial hashing. We have implemented the spatial hashing method described in [THM+03]. This method performs the worst due to hash collisions and a high memory transfer invoked by the hash function. Note that even when reordering the particles according to a Z-curve, the improvement is marginal since we traverse the whole hash table and spatially close cells are not close in memory. However, if we loop over the particles and not the whole grid, spatial compactness can be exploited. In this case, the query time reduces to 50ms when particles are reordered (this number is not given in Table 4.1).

Compact hashing. The proposed compact hashing exploits temporal coherence in the construction step. Thereby, the insertion of particles into the grid is five times faster than for spatial hashing. Furthermore, the query is nearly three times faster due to the reduced memory transfer invoked by the compact list of used cells and the hash collision flag.

Reordering. Note that we reorder the particles every 100th step according to a Z-curve in all methods except index sort. By reordering the particles according to a Z-curve, spatially close particles reside close in memory. Thus, particles that are close in memory are very likely spatial neighbors. Since in SPH, most computations are interpolations requiring informations of neighbors, the Z-curve increases the cache-hit rate for the neighborhood query and for the fluid update. Reordering significantly reduces the memory transfer and, thereby, improves the performance (see Table 4.2). Please note that for faster sorting libraries, reordering more frequently could be beneficial.

We summarize that compact hashing is as efficient as Z-index sort. Both methods outperform spatial hashing and the basic uniform grid. For compact hashing, the memory consumption scales with the number of particles, while for Z-index sort it scales with the

cell size	pairs	tested pairs	ratio	query
h	3.6M	25M	6.9	26.6
$0.5h$	3.6M	15.5M	4.3	39.6

Table 4.3: Influence of the cell size on the ratio of tested pairs to influencing pairs and the query time in millisecond. Test scene: CBD 130K.

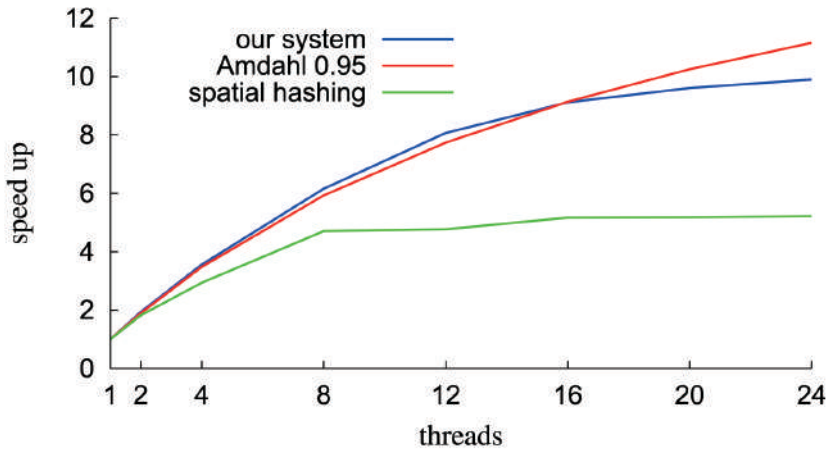

Figure 4.5: Total speed up of our system for fluid update and neighbor search using compact hashing. The scaling is in good agreement with Amdahl's law. For comparison, the scaling of spatial hashing is given.

domain. Therefore, for very large or unrestricted domains, compact hashing should be preferred. On the other hand, if influencing pairs can not be stored, Z-index sort should be used since the query time is lower.

4.3.2 Cell Size

The size of the spatial cells influences the number of potential pairs that have to be tested for interaction. Generally, smaller cell sizes approximate the shape of the influence radius h better which is a sphere in 3D. If the cell size d is chosen as h, only 27 cells need to be tested for interaction. If $d < h$, the number of potential cells grows, but less particles have to be tested for interaction. However, if more cells are queried, the memory transfer increases. We observed that the increased memory transfer is more expensive than testing more potential neighbors for interaction (see Table 4.3). Thus, the optimal cell size is the influence radius.

4.3.3 Parallel Scaling

Optimally, the speed up from parallelization would be linear. However, the optimal scaling can not be expected due to the parallelization overhead for synchronization and communication between different threads. According to Amdahl's law [Amd67], even a small portion of the problem which cannot be parallelized will limit the possible speed up. For example, if the sequential portion is 10%, the maximum speed up is 10.

CPUs	compact hashing	Z-index sort	PCISPH	SESPH
1	404.1	391.3	791.1	189.5
24	40.3	43.1	78.6	17.6

Table 4.4: Parallel scaling of neighbor search and simulation update. For 24 CPUs a speed up of 10 is achieved. Timings are in milliseconds for CBD 130K.

		neighbor	PCISPH			SESPH		
scene	# p	search [ms]	update [ms]	$\Delta t[s]$	ups	update [ms]	$\Delta t[s]$	ups
Glass	75K	30.8	29.8	0.0025	16.5	7.3	0.0001	26.2
CBD 130K	130K	40.3	78.6	0.0006	8.3	17.6	1.8e-5	16.7
CBD large	1.7M	427.8	1130.0	0.0002	0.6	271.6	5.7e-6	1.4
River	12M	5193.6	11941.8	0.0005	0.06	-	-	-

Table 4.5: Average performance in milliseconds for different scenes. Compact hashing is used in all scenes. Fluid update times, time step and simulation updates per lab second (ups) are given for PCISPH and for SESPH, for a similar level of compressibility.

Rewriting algorithms, in order to circumvent data dependencies is fundamental to increase the possible speed up. Furthermore, the performance of an algorithm is either memory or CPU bound. While for CPU-bound algorithms the performance is easily increased by using an additional or faster CPU, the bottleneck for memory-bound problems is the bandwidth. Therefore, only strategies that are reducing the memory transfer are improving the efficiency.

The presented strategies and techniques employed in Z-index sort and the compact hashing as well as the reordering of particles lower the memory transfer and, thus, improve the performance of the system significantly. The scaling and even the flattening of the proposed system is in very good agreement with Amdahl's law for a problem that is parallelized to 95% (see Figure 4.5 and Table 4.4). In contrast, the scaling of spatial hashing is much worse.

4.3.4 Scaling with Particles

Finally, we show that our system scales linearly with the number of particles (see Table 4.5). We therefore set up different small-scale and large-scale simulations. In the *glass scene*, a glass is filled with 75K particles (see Figure 4.6). For this scene, a plausible simulation result is achieved by PCISPH for a time step of 0.0025 seconds, where ten *real world* seconds could be simulated in four minutes. With SESPH we computed a similar result with a 25 times smaller time step in 63 minutes.

Due to the large memory capacity of CPU architectures, our system shows good performance for large-scale simulations with millions of particles. For those setups, querying the particles multiple times has a significant impact on the performance. On average, for the *CBD large scene*, 341 million pairs are queried for interaction per simulation step and 2.3 billion pairs for the *river scene* (see Figure 4.7 and Figure 4.8).

Our system updates an SESPH simulation of 130K particles 17 times per second. This performance is similar to the GPU implementation presented in [GSSP10].

Figure 4.6: Glass scene. The 'wine' is sampled with up to 75K particles.

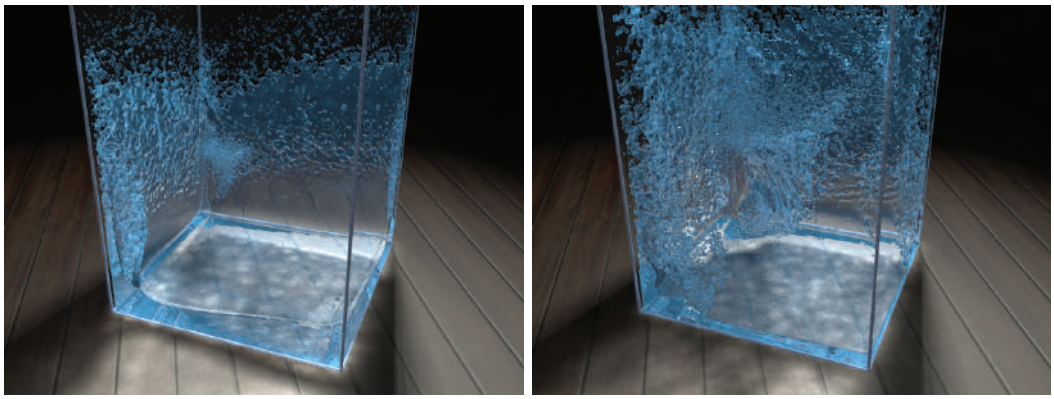

Figure 4.7: CBD large scene. A corner breaking dam with 1.7 million particles simulated with PCISPH.

4.4 Discussion

This chapter has discussed important aspects which are critical for the performance of a parallel SPH implementation. Data structures for an efficient neighborhood search have been proposed. Construction and query times are reduced by lowering the memory transfer. This is achieved by mapping spatial locality onto memory via a space-filling Z-curve. Performance aspects and the scaling on parallel architectures have been thoroughly analyzed.

The proposed structures do also scale well on the GPU. They have been successfully employed to speed up SPH simulations on the GPU, e.g., [OK12, MM13]. Furthermore, in [AIAT12], the presented Z-index sort method has been applied to speed up the surface reconstruction of SPH simulations on CPU and GPU.

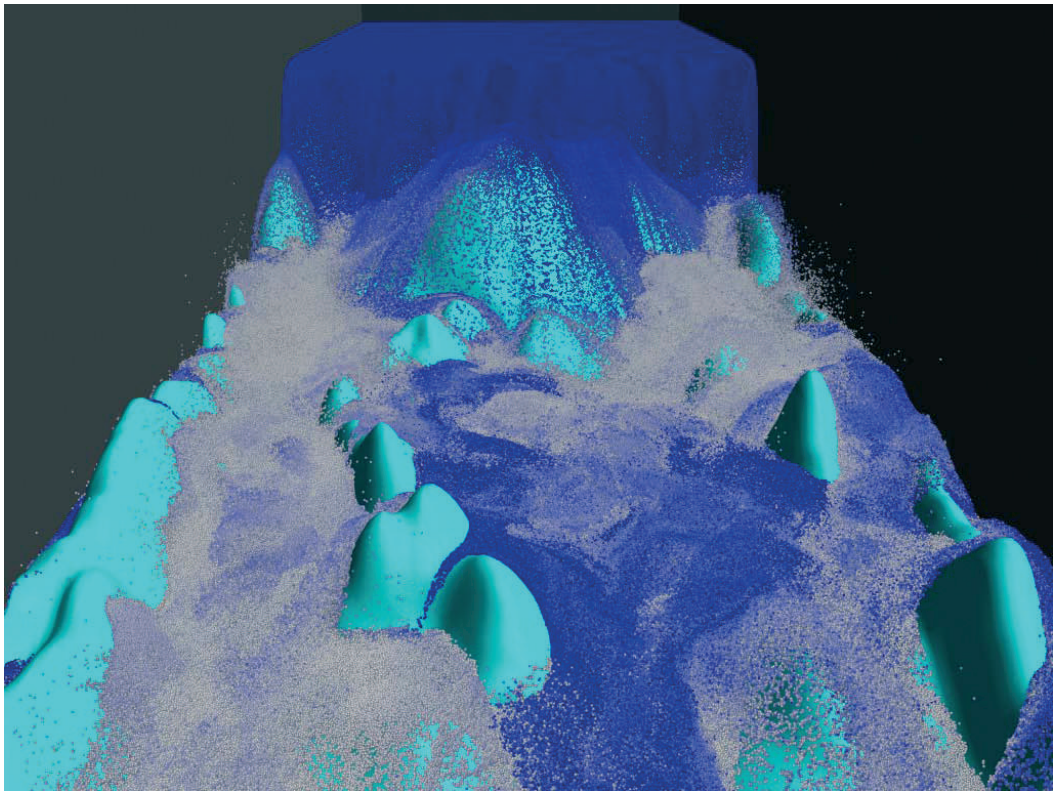

Figure 4.8: River scene. The fluid consists of 12.1 million particles and the terrain is sampled with more than 5 million particles. The particles are color-coded according to acceleration.

CHAPTER 5

Boundary Handling

The interaction of the fluid with rigid boundaries requires special consideration in order to prevent penetration of objects. In this chapter, a new boundary scheme is proposed which incorporates the boundary into the pressure system of the fluid. Similar to [BTT09], particle positions are predicted and corrected in order to guarantee *watertight* boundaries. However, by taking the current pressure on the boundary into account, smoother density distributions at fluid-boundary interfaces are achieved and stacking of particles is prevented. For the proposed scheme, larger time steps can be used compared to existing approaches while parameter tweaking is not required.

This chapter is structured as follows: Section 5.1 reviews existing boundary-handling methods for SPH. The proposed pressure-based coupling is explained in Section 5.2. Extensions to handle arbitrary objects and to realize two-way coupling are given in Section 5.3 and Section 5.4, respectively. The effectiveness of the approach is analyzed in Section 5.5 and discussed in Section 5.6.

5.1 Related Work

In most SPH simulations, boundary conditions are enforced using **penalty forces** that scale with the distance of the fluid particle to the boundary. [Mon94] sampled the boundary with particles which exert central Lennard-Jones penalty forces. For this formulation, the boundary force scales polynomially with the distance of the fluid particle. This causes large pressure variations in the fluid. Thus, the time step has to be chosen very small for weakly compressible fluids. Müller et al. [MST+04] adapted the penalty-based forces to simulate the interaction of SPH fluids with particle-sampled deformable meshes. They achieved very realistic two-way coupling by using a smooth formulation of the Lennard-Jones forces. Later, Monaghan [Mon05, MK09] employed a more stable form of the repulsion forces using a scaled version of the cubic spline kernel. But still, the main difficulty in pure penalty-based approaches is controlling the stiffness parameter. This parameter has to be balanced such that boundary penetration is avoided, while not causing pressures that are too high. Therefore, small time steps are required to produce smooth pressure distributions. Unfortunately, the stiffness parameter and, consequently, the time step must be chosen carefully for each scenario.

The above-mentioned boundary handling methods do not control the particle position after a collision directly. Therefore, high pressure forces might be introduced if the repulsion force is too strong. In order to overcome this problem and to have more control on the boundary condition, direct-forcing approaches were applied. In [BTT09], one- and two-way

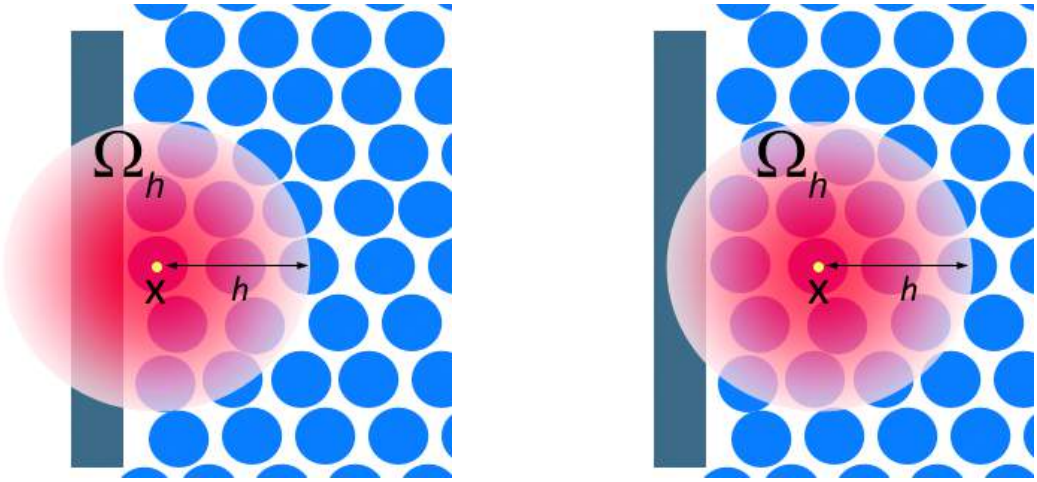

Figure 5.1: Particle deficiency at the solid boundary results in poor approximations of field variables as the interpolation domain is not sufficiently sampled. This affects not only particles directly colliding with boundaries (left), but all fluid particles for which the solid boundary is in the support domain (right).

coupling of rigid bodies and fluids were realized by computing control forces and velocities using a predictor-corrector scheme. This method can simulate different slip-conditions. Furthermore, non-penetration is guaranteed for each time step which prevents large (penalty) forces. Therefore, larger time steps can be used in this method.

A phenomenon that occurs in distance-based penalty schemes and in direct forcing is the sticking of fluid particles to the solid boundary. This is a result of a *particle deficiency* at the interface. In other words, at the interface with the free surface or solid boundary, the support domain is not sufficiently sampled and, thus, field variables can not be well approximated with the SPH interpolation concept. Harada et al. addressed this problem for the distance-based approach [HKK07b]. In this approach, the control force constrains the fluid particle positions to the boundary surface if formulated in a predictive-corrective way similar to [BTT09]. In order to avoid sticking artifacts, a wall-weight function is employed which adds a weighted contribution of the boundary to the fluid density. This value is precomputed and depends only on the distance of the fluid particle to the boundary. As shown in this chapter, the stacking of particles is reduced in this approach. However, it also suffers from irregular density distributions at the boundary which might lead to unnatural accelerations.

In order to achieve a smooth reconstruction of the field quantities at solid boundaries, two alternative methods have been proposed, namely the ghost-particle and the frozen-particle method. In the ghost-particle method, fluid particles close to the boundary are mirrored across the boundary. A ghost particle gets the same pressure, viscosity, mass and density of its corresponding fluid particle while the normal component of the velocity is inverted. This method was successfully employed to simulate different slip conditions for straight [HA06] and curved [MM97, SB12] boundary surfaces. Ghost particles are generated on the fly which can be challenging, e.g., for complex boundaries, ghost masses might be erroneously introduced in sharp regions. In [FG07], fixed predefined mirror particles are proposed in order to handle sharp regions. Frozen particles are fluid particles which are non-moving for static boundaries, while for rigid bodies they are restricted to rigid-body motion [SSP07, KAD+06]. Sampling the boundaries with ghost or frozen particles is effective

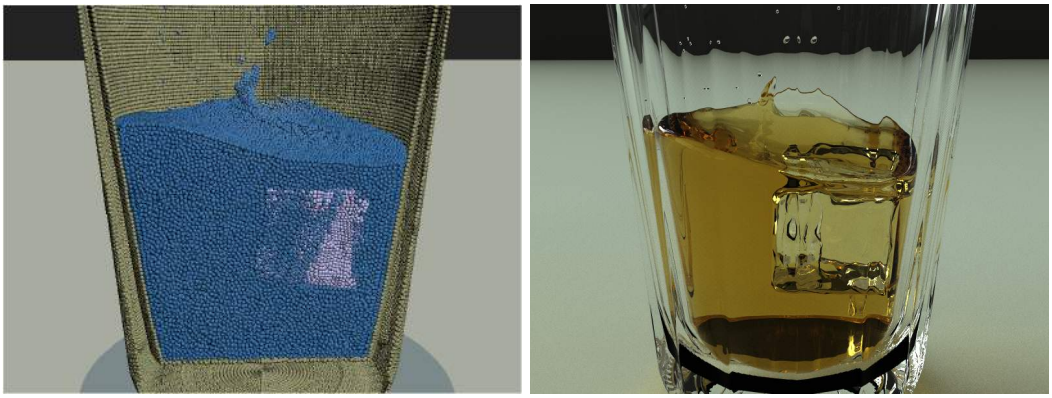

Figure 5.2: The proposed boundary method reconstructs the field variables at the solid-fluid interface smoothly by incorporating the boundary into the SPH interpolation. Therefore, solid boundaries are sampled with particles. A clipped view of the simulation (left) shows the particle discretization of the fluid volume (blue), the glass (green) and the ice cube (purple). The corresponding frame rendered with mental ray is shown on the right.

for obtaining a better approximation of the field variables at the solid-interface. However, the numerical stability and the quality of the simulation depends on the sampling of the boundaries. Simple objects like a plane or a cube can be equidistantly sampled with particles, but for complex shapes with convex and concave regions maintaining equal distances is nearly impossible. Furthermore, sampling only the surface of solid objects with a single layer might not be sufficient in order to guarantee non-penetration [DK01]. However, multiple layers of boundary particles do not only affect the performance, but also prohibit the interaction of the fluid with thin objects.

5.2 Pressure-based Coupling

The approximation of the density plays a central role in SPH-based fluid simulations as nearly all solvers rely on the density in order to evaluate pressure. Due to the SPH concept, the density computation simplifies to a summation of weighted mass-contributions of particles in the support domain, see (2.49). This is often referred to as the density summation approach. However, at interfaces, the support domain might not be sufficiently sampled which leads to an underestimation of densities near the boundaries as illustrated in Figure 5.1. In order to alleviate this underestimation, the Shepard filter [Pan04] can be employed. Although this iterative correction scheme significantly improves the estimate, fluid particles close to an interface still do not have neighbors on the other side of the material which could apply any force. Consequently, the motion of these particles is constrained to the boundaries which causes particles to stick to the solid boundary.

In order to avoid this problem, we take the neighboring boundary particles into account when computing densities and forces for fluid particles. As the focus of this thesis is set on efficiency, we aim for a minimal amount of boundary particles in order to save memory and to gain performance. Therefore, we sample only the surface of the rigid objects using a single layer of particles. The particle representations of rigid bodies in the framework are computed directly for analytical shapes. For triangle meshes, particles are generated based on [BYM05] which yields a quite homogenous sampling as illustrated in Figure 5.2.

Each boundary particle b stores its position $\mathbf{x}_b$, normalized normal $\mathbf{n}_b$, mass m_b, density ρ_b and pressure p_b. In the following, we assume that the spacing of the boundary particle is equal to the equilibrium distance $r_0 = 0.5 \cdot h$ of the fluid particles. Thus, a fluid particle i is considered to penetrate the boundary at position $\mathbf{x}_b$, if $\|\mathbf{x}_i - \mathbf{x}_b\| < r_0$. Boundary particles b contribute to the densities of fluid particles i as

$$\rho_i(t) = \sum_j m_j W(\mathbf{x}_{ij}, h) + \sum_b m_b W(\mathbf{x}_{ib}, h), \tag{5.1}$$

where we use the same support radius h for fluid and boundary particles. The mass of boundary particles is set to the same value as for the fluid. In order to prevent fluid particles to penetrate the solid object, boundary particles apply pressure forces onto the fluid. Thus, the pressure force is rewritten as

$$\begin{aligned}\mathbf{F}_i^{pressure}(t) = &-m_i \sum_j m_j \left(\frac{p_i(t)}{\rho_i^2(t)} + \frac{p_j(t)}{\rho_j^2(t)} \right) \nabla W_{ij}(t) \\ &- m_i \sum_b m_b \left(\frac{p_i(t)}{\rho_i^2(t)} + \frac{p_b(t)}{\rho_b^2(t)} \right) \nabla W_{ib}(t),\end{aligned} \tag{5.2}$$

where the pressure of boundary particles is computed in the same way as for fluid particles. Accordingly, the solid surface acts as an interface which is treated as if it belongs to the fluid. Since the pressure of the boundary particles increases with the surrounding fluid, density particles and boundary particles do apply pressure forces onto the fluid. Thereby, sticking of fluid particles to the boundary is counteracted by the proposed method.

Recall that for performance reasons, we only sample the surface of rigid objects with just a single layer of boundary particles. Therefore, for fluid particles that are directly interacting with the boundary, the support domain might still not be fully sampled, see Figure 5.1, left. As such missing samples would have at least a distance of $r_0 + \|\mathbf{x}_i - \mathbf{x}_b\|$, the contribution of these missing samples can be expected to be close to zero due to the Gaussian form of the kernel function. Nevertheless, we observed that in some scenarios, the computed boundary pressure forces might not be sufficient to prevent particles from penetrating the boundary. Therefore, we propose to additionally correct such penetrations after the forces have been computed.

According to the boundary sampling, it is possible that a fluid particle i penetrates more than one boundary particle at the same time. Thus, for concave objects with sharp features, iteratively correcting the positions as in [BTT09] might lead to time-inconsistent corrections of the particle positions. We therefore propose to estimate the penetration depth and direction by employing a weighting function to compute a time-consistent correction force $\mathbf{F}_i^b$ in the sense of [HTK+04]. Accordingly, for each fluid particle i that penetrates a solid boundary, we first compute an average boundary normal $\mathbf{n}_i^c$ of all boundary particles b which are penetrated by particle i as

$$\mathbf{n}_i^c = \sum_b w_{ib}^c \mathbf{n}_b \tag{5.3}$$

$$w_{ib}^c = \max\left(0, \frac{r_0 - \|\mathbf{x}_{ib}^*\|}{r_0}\right), \tag{5.4}$$

where $\|\mathbf{x}_{ib}^*\| = \|(\mathbf{x}_i^*(t + \Delta t) - \mathbf{x}_b)\|$.

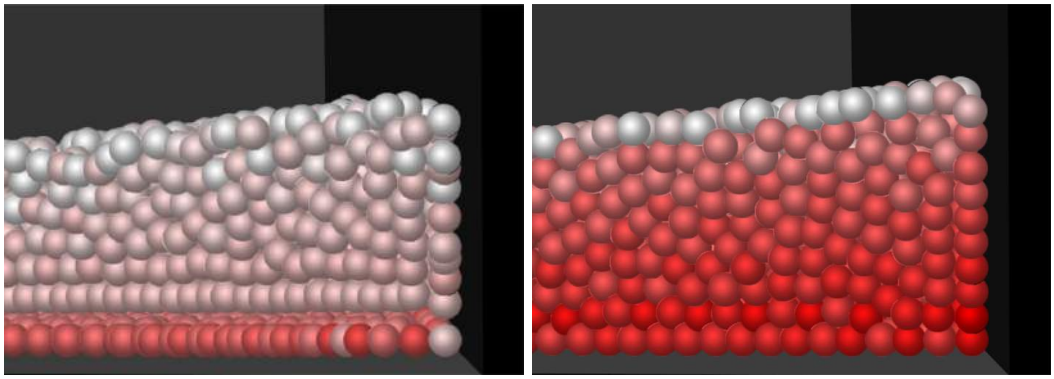

Figure 5.3: Influence of the boundary method on the pressure distribution. Pressure is colored coded where red is maximum. In [BTT09] (left), the density of fluid particles is not well reconstructed causing particles to cluster at the solid interface. This causes high density ratios and large pressure gradients. The proposed method, incorporates boundaries into the interpolation and accounts for the pressure on the boundary. Thereby, a smooth distribution is achieved (right).

The position of the particle i is then corrected according to $\mathbf{n}_i^c$ with

$$\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i^*(t+\Delta t) + \frac{1}{\sum_b w_{ib}^c} \sum_b w_{ib}^c (r_0 - \|\mathbf{x}_{ib}^*\|) \frac{\mathbf{n}_i^c}{\|\mathbf{n}_i^c\|}. \tag{5.5}$$

The resulting velocity is computed as

$$\mathbf{v}_i(t+\Delta t) = \epsilon \left[\mathbf{v}_i^*(t+\Delta t)\right]_t, \tag{5.6}$$

where $\left[\mathbf{v}_i^*(t+\Delta t)\right]_t = \mathbf{v}_i^*(t+\Delta t) - \left[\mathbf{v}_i^*(t+\Delta t)\right]_n$ denotes the tangential velocity and $\left[\mathbf{v}_i(t)\right]_n = (\mathbf{v}_i(t) \cdot \mathbf{n}_b) \cdot \mathbf{n}_b$ the normal velocity. The friction can be controlled with $\epsilon \in [0,1]$.

Please note that in [BTT09], the position update leads to higher density ratios in the fluid at the boundary interface (see Figure 5.3). Thus, smaller time steps might be required in order to enforce incompressibility. Furthermore, sticking of particles might occur in boundary regions since the normal velocities are fully damped in [BTT09]. In order to prevent this behavior, the coefficient of restitution might be chosen larger than zero. However, this can lead to rigid-body-like collisions for the fluid particles. In the proposed model, these drawbacks of the direct-forcing approach are eliminated by taking the density contribution of the boundary particles and, hence, the pressure at the boundary into account.

5.3 Volume Correction

Until now, we have assumed that the rigid is uniformly sampled, and thus, we have set the volume V_b of all boundary particles equal to $V_i = m_i/\rho_0$ where i denotes a fluid particle with rest density ρ_0. However, for irregular samplings of solid meshes this assumption is not valid and might cause instabilities. For non-homogeneous samplings, the fluid density is overestimated in regions where the rigid is sampled more densely and underestimated in regions where the sampling is coarse. The falsified densities induce wrong pressure values which might result in numerical instabilities at the interface. This issue is addressed in [AIS+12], where the presented method is extended to two-way interactions of rigid objects with the

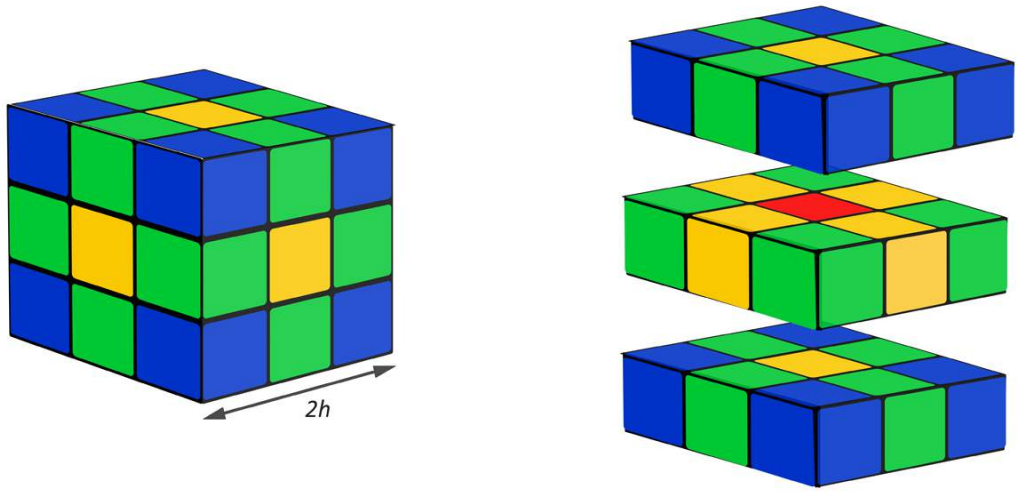

Figure 5.4: Distribution of kernel weights for cubical sub-volumes in 3D. The point of interest is at the center of the cube. Kernel weights are color coded. Red represents 31.8%, yellow 8%, green 1.6%, blue 0.15% of the total kernel weight.

fluid. While a detailed discussion of this approach is beyond the scope of this thesis, the following describes the key concept of the approach as we will use the method for realizing two-way coupling with IISPH.

The idea of the proposed boundary handling is that a boundary particle contributes to the interpolation of quantities in the same way as a fluid particle. In order to account for the sampling, the contribution should be proportional to the volume ratio of the solid particle and the fluid particle. In other words, rigid particles of densely sampled regions should contribute less than particles of coarsely sampled regions. Thereby, the volume contribution of the regions should be the same. The volume ratio of a boundary particle b to a fluid particle i is given as

$$\frac{V_b}{V_i} = \frac{m_b \rho_0}{m_i m_b \delta_b^{rigid}} = \frac{\rho_0}{m_i \delta_b^{rigid}}, \tag{5.7}$$

where δ_b^{rigid} denotes the *number density* of a boundary particle b with respect to all rigid particles within distance h. δ_b^{rigid} can be computed with

$$\delta_b^{rigid} = \sum_{j \in rigid} W(\mathbf{x}_{bj}, h). \tag{5.8}$$

In general, we can assume that the boundary-particle neighborhood lies on a 2D space as we only sample rigid objects along the surface. E.g., for a uniformly sampled plane with particle spacing r_0, a prototype boundary particle has on average nine neighbors, including itself. Figure 5.4 illustrates how the partial volume integrals and, thus, the kernel weights are distributed in the support domain. As can be seen, for our prototype particle, 70% of the volume integral is sampled with boundary particles. In this case, $m_i \delta_b^{rigid}$ computes to $0.7\rho_0$ independent of the choice of h. However, if the sampling gets more dense, δ_b^{rigid} increases.

There are now various possibilities to account for the sampling in order to enforce continuous contributions of the boundary. In [AIS+12], the following function is proposed to adjust the

contributions:

$$\Psi_b(\rho_0) = \frac{\rho_0}{\delta_b^{rigid}}. \tag{5.9}$$

Note that in the example of a uniformly sampled solid plane, (5.9) does not enforce a volume ratio V_b/V_i of 1, but of approximately 1.43. The motivation here is to account for the missing layer of boundary particles. In order to enforce a volume ratio of 1, $\Psi_b(\rho_0)$ should be multiplied by 0.7.

5.4 Implementation

The pressure-based coupling has been incorporated into PCISPH in [IAGT10] and in [AIS+12]. For the sake of completeness, we now demonstrate how to incorporate the method into IISPH. In order to realize two-way coupling with complex-shaped objects, we use the *sampling-aware* scheme and the forces presented in [AIS+12]. Therefore, the density estimation (3.16) is extended to

$$\begin{aligned}\rho_i(t+\Delta t) = &\sum_j m_j W_{ij} + \sum_b \Psi_b(\rho_{0_i}) W_{ib} \\ &+ \Delta t \sum_j m_j \mathbf{v}_{ij}(t+\Delta t)\nabla W_{ij} \\ &+ \Delta t \sum_b \Psi_b(\rho_{0_i}) \mathbf{v}_{ib}(t+\Delta t)\nabla W_{ib}.\end{aligned} \tag{5.10}$$

For a weak two-way coupling, we assume a constant rigid velocity $\mathbf{v}_b$ throughout the pressure iterations. Therefore, we can estimate the density without pressure forces as

$$\begin{aligned}\rho_i^{adv} = &\sum_j m_j W_{ij} + \sum_b \Psi_b(\rho_{0_i}) W_{ib} \\ &+ \Delta t \sum_j m_j \mathbf{v}_{ij}^{adv} \nabla W_{ij} \\ &+ \Delta t \sum_b \Psi_b(\rho_{0_i}) \left(\mathbf{v}_i^{adv} - \mathbf{v}_b(t+\Delta t)\right) \nabla W_{ib}.\end{aligned}$$

Pressure forces correct the density field such that

$$\begin{aligned}\rho_i(t+\Delta t) = \rho_i^{adv} &+ \sum_j m_j \left(\Delta t^2 \frac{\mathbf{F}_i^p}{m_i} - \Delta t^2 \frac{\mathbf{F}_j^p}{m_j} \right) \nabla W_{ij} \\ &+ \sum_b \Psi_b(\rho_{0_i}) \left(\Delta t^2 \frac{\mathbf{F}_i^p}{m_i} \right) \nabla W_{ib}.\end{aligned}$$

In [AIS+12], boundary particles b do not have an individual pressure, but exert a pressure force which is dependent on the fluid pressure p_i as

$$\mathbf{F}_{i \leftarrow b}^p = -m_i \Psi_b(\rho_{0_i}) \frac{p_i}{\rho_i^2} \nabla W_{ib}. \tag{5.11}$$

Thus, the displacement due to pressure is computed as

$$\Delta t^2 \frac{\mathbf{F}_i^p}{m_i} = \sum_j \underbrace{-\Delta t^2 \frac{m_j}{\rho_j^2} \nabla W_{ij}}_{\mathbf{d}_{ij}} p_j + \underbrace{\left(-\Delta t^2 \sum_j \frac{m_j}{\rho_i^2} \nabla W_{ij} - \Delta t^2 \sum_b \Psi_b(\rho_{0_i}) \frac{1}{\rho_i^2} \nabla W_{ib} \right)}_{\mathbf{d}_{ii}} p_i.$$

Accordingly, the pressure update with relaxed Jacobi reads

$$\begin{aligned} p_i^{l+1} = (1-\omega) p_i^l + \omega \frac{1}{a_{ii}} \Bigg(& \rho_0 - \rho_i^{adv} \\ & - \sum_j m_j \Big(\sum_j \mathbf{d}_{ij} p_j^l - \mathbf{d}_{jj} p_j^l - \sum_{k \neq i} \mathbf{d}_{jk} p_k^l \Big) \nabla W_{ij} \\ & - \sum_b \Psi_b(\rho_{0_i}) \sum_j \mathbf{d}_{ij} p_j^l \nabla W_{ib} \Bigg). \end{aligned} \tag{5.12}$$

5.5 Results

In order to show the benefits of the presented boundary scheme, we compare it with the direct-forcing methods [BTT09] and [HKK07b]. The methods are tested for a simple corner breaking dam scene with 23k particles. For the comparisons, both reference methods are integrated into the PCISPH algorithm (see Algorithm 2).

We compare the different schemes with respect to the influence on the density distribution and the time step. For all methods, the maximum time step is employed which satisfies an overall volume compression $\eta_{avg}^{\%}$ smaller than 1%. Since for the PCISPH method, these values are also influenced by the number of iterations required to correct the density error, we are using three iterations in all simulation steps. For all methods, we enforce a free-slip condition, i.e., the tangential velocity is not damped.

In Figure 5.3, a side-by-side comparison of [BTT09] and the proposed method is given. Particles are color coded according to the pressure distribution. The coloring scheme assigns the color red to the highest pressure and white to the lowest pressure in the current time step. These values are interpolated according to $\sqrt{\frac{p_i(t)}{p^{max}(t)}}$. Note that the maximum pressure and, hence, the color schemes are not normalized, i.e., for different methods, the same color might refer to different pressures.

As can be observed, the pressure distribution differs a lot for the three boundary methods. In [BTT09], the normal velocity of colliding fluid particles is fully damped. In the test scenario, the fluid volume covers the floor completely and therefore, particles on the floor get stuck. This results in a high pressure ratio of the particles 'colliding' with the boundary and their 'non-colliding' neighbors (on top). Due to high pressures of the colliding particles, a spatial gap between the particles arises (see Figure 5.3, left). Furthermore, the high pressure ratio causes unnatural acceleration.

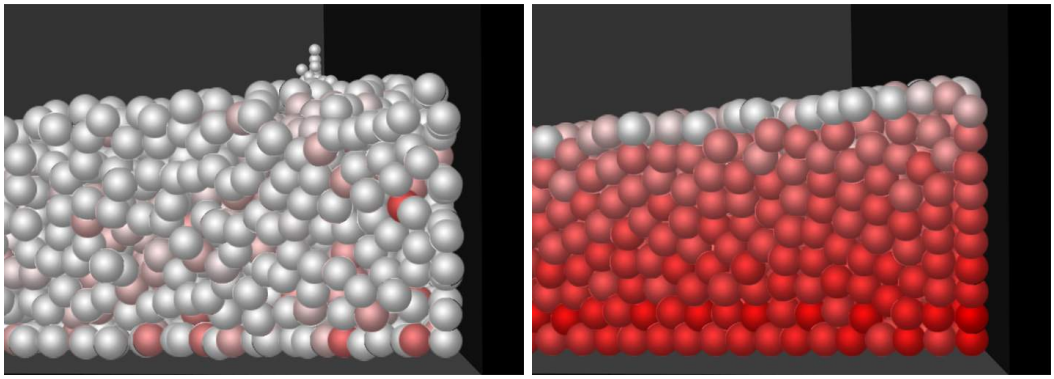

Figure 5.5: Influence of the boundary method on the pressure distribution. Pressure is colored coded. Left, noisy distribution caused by the distance-based wall weight function employed in [HKK07b]. Right, smooth distribution achieved by the proposed method (right).

In [HKK07b], the boundary contributes to the density of the fluid particles. A distance-based control force is computed via a wall-weight function. This value is pre-computed and depends only on the distance of the fluid particle to the boundary. The local fluid sampling is not taken into account. This results in a noisy pressure distribution and high pressure ratios which is illustrated in Figure 5.5, left.

Large pressure ratios result in strong pressure forces and, hence, high accelerations. In contrast to [HKK07b, BTT09], the proposed method computes the pressure on the boundary particles like for fluid particles. Therefore, particles do not get stuck on the boundary, while the pressure distribution is very smooth as illustrated in Figure 5.5, right. Thus, our method can generally handle larger time steps compared to [HKK07b, BTT09]. For the test scene, the following time steps could be used: $\Delta t = 0.0008$ [BTT09], $\Delta t = 0.0015$ [HKK07b] and $\Delta t = 0.0045$ for the proposed boundary method.

Note that in penalty-based boundary methods, large forces and, thus, small time steps are required to resolve the penetration into the boundary. Since the proposed method enforces non-penetration by incorporating boundary particles into the pressure system, much larger time steps can be used.

5.6 Discussion

In this chapter, a new method for handling the fluid-solid interface in SPH is presented. The incorporation of boundary particles into the computation of density and pressure forces eliminates the drawbacks of penalty-based methods. Incorporated into an incompressible SPH solver, smooth density distributions and pressure distributions are enforced. Non-penetrations of complex shaped boundaries is enforced even for comparatively large time steps. As only the surface of rigid objects needs to be sampled, the computational overhead and memory consumption is kept small. Finally, it has been shown how two-way coupling with rigid objects can be realized in IISPH.

CHAPTER 6

Multi-phase Fluids

In the previous chapters, SPH approximations for compressible and incompressible fluid flows have been employed for simulations of a single fluid. As the given formulations already consider phase-specific quantities such as mass or volume, the presented algorithms can be directly employed for simulations of multi-phase flow, at least theoretically. In practice however, immiscible fluids with a density ratio larger than 10 can not be realistically simulated if the standard SPH density summation is used. As discussed in [SP08], the reason is that in SPH, the macroscopic flow is mainly governed by the density computation. Over- or underestimating the density leads to erroneous pressure values which might result in unnatural acceleration caused by erroneously introduced pressure ratios. In [SP08], a different density model is proposed which treats all particle neighbors as if they belong to the same phase, i.e., have the same mass and rest density. Thereby, the densities at the interface are computed correctly.

Although this method can represent sharp density changes at the interface, it suffers from severe limitations when dealing with large density ratios. The buoyancy of small, light volumes is significantly damped. As stated in [SP08], this is due to the pressure force which compels the particles to arrange in a stable lattice structure. This structure can not be broken by small volumes such as air bubbles.

Air bubbles are a natural phenomenon that occurs in everyday life. Whenever a liquid is poured into a glass, bubbles are created by trapped air. Accordingly, the visual realism of fluid animations is significantly enhanced by modeling the creation and flow of bubbles. In reality, air and water are interacting in a two-way manner. Unlike water droplets, air bubbles are under strong velocity diffusion because they are coupled to the surrounding fluid by drag and lift forces. According to the very large density ratio of air to water (≈ 1000), water exerts a high pressure on air bubbles which makes them merge rapidly. As the bubbles grow, they rise faster due to the rapid increase in buoyancy. In turn, those large and fast rising bubbles significantly influence the liquid flow. As stated in [SP08], modeling high density ratios with SPH is problematic and may lead to numerical instabilities due to large forces. The simulation of air bubbles is not possible with the standard SPH method [MSKG05] nor with the adapted multi-phase method [SP08].

In this chapter, a new SPH model for simulating air bubbles is presented. In order to handle the high density ratio of air and water, we treat the two phases separately. We account for the interaction of the two phases by employing a drag force. As we show, the proposed drag force is sufficient to capture the two-way interaction realistically, while the numerical stability is not affected. We model the buoyancy of the air bubbles by a saturated function that accounts for the volume. Thereby, larger bubbles rise faster than smaller bubbles. Furthermore, a cohesion force is employed that minimizes the surface and makes

rising bubbles merge. In order to simulate trapped air without explicitly modeling the air surrounding the fluid, we generate air particles on the fly in surface regions with high velocity differences. When air particles have reached the surface, they are treated as foam and finally deleted after a user-defined time. The presented bubble model can be easily incorporated into any existing SPH solver with negligible computational overhead.

6.1 Related Work

At the time of writing only few works have addressed the simulation of air bubbles in a pure Lagrangian framework. In [MSKG05], the standard SPH model for single-phase fluid simulations [MCG03] is extended to handle multiple fluids with density ratios of up to 10. In order to simulate air bubbles, an artificial buoyancy force is applied. However, as stated in [SP08], this approach suffers from falsified density estimates at the interface which induce wrong pressure values. This limits the time step and might result in numerical instabilities for fast rising air particles due to large pressure forces. [SP08] overcomes this problem by ignoring the mass in the computation of the particle density. Thereby, sharp density changes at the fluid interface can be reproduced. Although this method can handle density ratios of up to 100, the flow of small, light volumes like air bubbles can not be realistically handled. According to Solenthaler et al., the buoyant volumes can not break up the crystallized particle configuration formed by the pressure forces. In order to circumvent these problems, we ignore particle neighbors of other phases when computing the density. Thus, we treat each phase separately. The interaction of both phases is modeled via a drag force.

A similar idea is presented in [CPPK07] for simulating dynamic gas bubbles generated from gas dissolution. In this approach, each phase is computed separately, where the air bubbles are modeled by discrete entities with fixed shape and the liquid is computed with SPH. The bubbles are coupled to the liquid via a drag force while the influence of the bubbles onto the liquid is neglected. In contrast, our bubble model is based on SPH and the governing forces are computed differently. Furthermore, we couple the liquid and air phase in a two-way manner using a different formulation of the drag force.

Capturing the fine scale flow of bubbles with Eulerian methods requires very fine grid resolutions. As stated in [HLYK08], for numerical reasons, each bubble should at least occupy 3 nodes in each dimension. Although the computational overhead can be minimized by adaptively refining the grid using an octree [LGF04], the required grid spacing significantly restricts the time step. Consequently, pure grid-based methods like the regional level set method [ZYP06] are only suited to handle relatively large bubbles in comparison to the fluid volume. Alternatively, hybrid methods have been proposed [HK03, GH04], in which the bubbles are simulated by passive air particles that are advected according to the underlying grid. Similar to [KVG02], the particles are modeled as spheres which do not change their shape. By coupling the bubble particles to a low resolution grid, millions of air particles can be simulated efficiently as shown in [KSK10]. However, in these models the interaction of particles is often neglected. Therefore, the size and shape of air bubbles is not varying over time. In contrast, in the proposed model, the air bubbles can consist of many particles. The employed cohesion force minimizes the bubble surface and makes bubbles merge. Since we reconstruct the bubble surface from the particle positions as described in [SP08], the bubble shape is deformable.

A hybrid solver is also proposed in [HLYK08], where bubbles are simulated with SPH and coupled to a grid-based fluid solver. The two phases are coupled via the velocity field. However, due to the insufficient resolution of the underlying grid, the path instability of air

bubbles can not be simulated by this coupling. In [HLYK08], path instability is captured by adapting the vorticity confinement method [FSJ01, SRF05] and artificial velocity disturbance based on random numbers. In contrast, we model the liquid phase with SPH. Thus, fine scale turbulences in the liquid can be simulated. The proposed two-way coupling is velocity-based. Thereby, the bubble flow is significantly influenced by the velocity field of the liquid. Consequently, the path instability of bubbles can be realistically simulated without adding artificial disturbance.

In [TSS+07], a two-dimensional shallow water model is coupled to a particle-based bubble simulation. In order to capture three-dimensional effects, the shallow water model makes a number of simplifying assumptions. In particular, the fluid flow is only modeled around bubbles and only if bubbles are in the fluid. Thereby, the model is very efficient to compute, but some effects can not be modeled, e.g., inertia effects of the fluid.

The remainder of this chapter is organized as follows: Section 6.2.1 presents force equations for controlling the bubble flow and the interaction of the air phase with the liquid. Furthermore, effective criteria for generating and deleting air bubbles are given. In Section 6.3, we show how this model can be integrated into the IISPH algorithm. Finally, the capability of the approach to animate bubble flow is analyzed in Section 6.4.

6.2 Air-Water Interaction

In order to simulate bubbles with SPH, we propose to simulate the air and liquid phase separately, i.e., only particle neighbors of the same phase contribute to the density, pressure and general force computations. Accordingly, problems like, e.g., high pressure ratios or buoyancy dampening do not occur. We account for the interaction of both phases by employing a new velocity-based coupling.

6.2.1 Bubble Physics

In general, bubbles have a sphere-like shape according to cohesion forces (interface tension). However, since bubbles are heavily influenced by the velocity field of the surrounding fluid, the bubble shape is deforming. Furthermore, bubbles are different in size, where larger bubbles rise faster and attract smaller bubbles.

For modeling these effects, we employ two forces that are computed for the air phase only, namely the buoyancy force $\mathbf{F}^{buoyancy}$ and the cohesion force $\mathbf{F}^{cohesion}$. The coupling of the two phases is realized by the drag force $\mathbf{F}^{drag}$.

Buoyancy. The buoyancy force accelerates air bubbles in opposite direction of the gravity force. In order to make larger bubbles rise faster than smaller ones, the buoyancy force should account for the volume of the air bubble V_{bub}. This leads to

$$\mathbf{F}_i^{buoyancy} = -k_b \cdot V_{bub} \cdot \mathbf{g}, \tag{6.1}$$

where k_b controls the buoyancy. However, computing V_{bub} is not straightforward without knowing which particle belongs to which bubble. The determination thereof invokes a significant computational overhead which can be avoided by using a heuristic formulation.

Replacing V_{bub} in (6.1) with $V_i = \frac{m_i}{\rho_i} = \frac{1}{\sum_j W(\mathbf{x}_{ij},h)}$ is not suitable since V_i gets smaller as the number of neighbors grows, i.e., an isolated particle would rise faster than a large

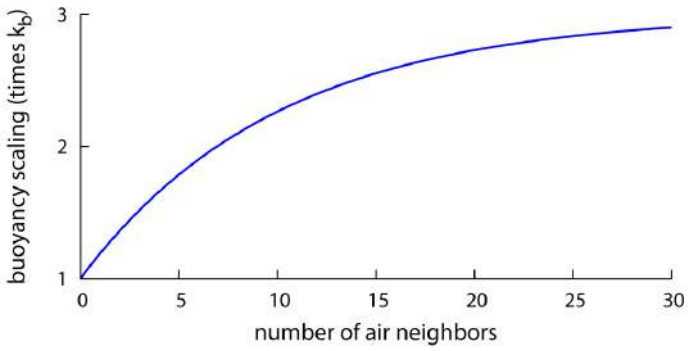

Figure 6.1: Scaling of the proposed volume-dependent buoyancy for $k_{max} = 3$. In order to account for the unknown total bubble volume, the buoyancy of an air particle is non-linearly related to its number of air neighbors.

bubble. On the other hand, correlating the buoyancy with the density ρ_i is also not optimal since, thereby, a compressed bubble (smaller volume) would rise faster than an expanded one.

Therefore, we propose to relate the magnitude of the buoyancy force to the number of particle neighbors n with

$$\mathbf{F}_i^{buoyancy} = -m_i k_b \cdot \left(k_{max} - (k_{max} - 1) \cdot e^{-0.1 n_i}\right) \cdot \mathbf{g}, \tag{6.2}$$

where k_b controls the minimum buoyancy and k_{max} the maximum buoyancy. The mass is added to the force formulation, in order to make the resulting acceleration independent of the simulation resolution. Thus, the resulting acceleration caused by the buoyancy force can be perfectly controlled since $\|k_b \mathbf{g}\| \leq \left\|\mathbf{a}_i^{buoyancy}\right\| \leq \|k_{max} \cdot k_b \cdot \mathbf{g}\|$. This is also illustrated in Figure 6.1.

Cohesion. The coalescence of air bubbles is an important effect. Smaller air bubbles are attracted by surrounding bubbles due to interface and surface-tension forces. We model this behavior by employing an artificial cohesion force

$$\mathbf{F}_i^{cohesion} = -k_c m_i \sum_j \rho_j \mathbf{x}_{ij}, \tag{6.3}$$

where k_c controls the strength of the cohesion force. According to (6.3), air particles are attracted by neighboring air particles with higher density. Thereby, spatially close air bubbles do merge. Note that the pressure force counteracts the attraction when the density ρ_i becomes too high. As a result, the surface of the bubble is minimized while the forces converge to an equilibrium.

Two-way coupling. In [CPPK07] and [HLYK08], the air phase is coupled to the liquid phase via an empirical drag force. The effect of the bubble momentum on the liquid phase is neglected. With respect to the generally high Reynolds number for air bubbles, the drag force $\mathbf{F}_i^{drag}$ acting on an air particle i is computed as

$$\mathbf{F}_i^{drag} = -k_d A_{bub} \sum_{liq} (\mathbf{v}_i - \mathbf{v}_{liq}) \left\|\mathbf{v}_{air} - \mathbf{v}_{liq}\right\|, \tag{6.4}$$

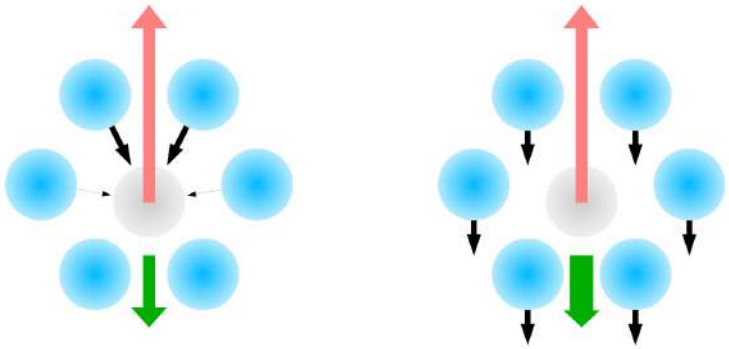

Figure 6.2: Comparison of the proposed drag force (left) and the drag force used in [CPPK07] (right). The influence of the liquid particles (blue) onto the air particle (grey) is shown. The velocity of the air particle is indicated by the red arrow, the liquid is at rest. Partial forces are illustrated by black arrows, while the resultant force is given by the green arrow. The thickness denotes the magnitude.

where k_d is a constant drag coefficient, liq the liquid neighbors of i and A_{bub} is the surface area of the bubble. Note that in [CPPK07], air bubbles are simulated as discrete spheres. Thus, A_{bub} is easy to compute. For the presented model it is hard to compute A_{bub} since bubbles may have arbitrary shapes and can consist of many particles.

In (6.4), the forces exerted by neighboring liquid particles are related to the velocity difference, but the flow direction and the distance of the particles are neglected. Thus, a liquid neighbor j that is very close, i.e., $\left\|\mathbf{x}_{ij} \leq \frac{h}{2}\right\|$, influences the velocity of the air particle by the same amount as a particle with distance h. Furthermore, the partial drag force $\mathbf{F}_{ij}^{drag}$ is non-zero as long as $\|\mathbf{v}_{ij}\| > 0$, whether the particles move towards each other or not.

In contrast, we account for the distance and flow direction (see Figure 6.2). Therefore, we propose a drag force that is motivated by the viscosity force and which couples both phases in a two-way manner. Consequently, the movement of the bubbles influences the velocity field of the fluid and vice versa. The drag force is defined as

$$\mathbf{F}_i^{drag} = m_i \sum_j m_j \frac{k_d h c_s}{\rho_i + \rho_j} \Pi_{ij} \nabla_i W(\mathbf{x}_{ij}, h), \tag{6.5}$$

where k_d denotes the drag constant. Π_{ij} is zero if either i and j belong to the same phase or if $\mathbf{v}_{ij} \cdot \mathbf{x}_{ij} \leq 0$, otherwise $\Pi_{ij} = \left(\frac{\mathbf{v}_{ij} \cdot \mathbf{x}_{ij}}{\|\mathbf{x}_{ij} + \epsilon h^2\|}\right)$.

Acceleration. The acceleration of an air particle is finally computed as

$$\mathbf{a}_{air} = m_{air}^{-1} \left(\mathbf{F}_{air}^{pressure} + \mathbf{F}_{air}^{cohesion} + \mathbf{F}_{air}^{buoyancy} + \mathbf{F}_{air}^{drag}\right) + \mathbf{g}, \tag{6.6}$$

while for a liquid particle liq, the acceleration computes to

$$\mathbf{a}_{liq} = m_{liq}^{-1} \left(\mathbf{F}_{liq}^{pressure} + \mathbf{F}_{liq}^{viscosity} + \mathbf{F}_{liq}^{drag}\right) + \mathbf{g}. \tag{6.7}$$

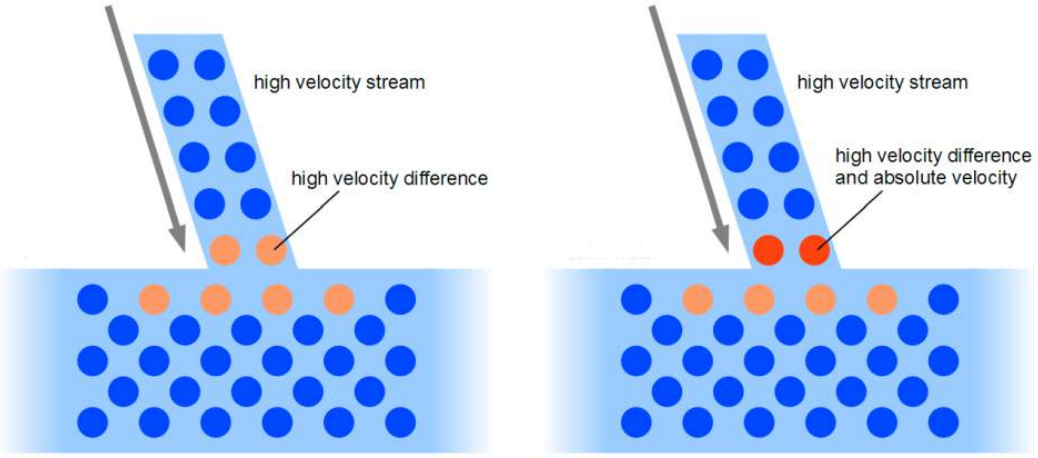

Figure 6.3: Air is dragged under water if a stream with high velocity hits the calm surface. Air particles are generated in regions with high velocity differences (orange) and a high absolute velocity (red).

6.2.2 Generation and Deletion of Air Particles

Air bubbles are generated when air is trapped inside the liquid. One possibility to capture this effect is to simulate the air phase surrounding the liquid explicitly. In order to avoid the implied computational overhead, we use a heuristic in order to detect regions where air is trapped.

The proposed heuristic is based on the following observations:

- Air molecules are pulled inside the liquid by inflows.
- The amount of trapped air grows with the velocity of the inflow.

Accordingly, we generate air particles in surface regions of high velocity differences. Therefore, for each liquid particle on the surface, the magnitude of the velocity difference $\mathbf{v}_i^{diff} = \sum_j \frac{m_j}{\rho_j}(\mathbf{v}_i - \mathbf{v}_j)W(\mathbf{x}_{ij}, h)$ is compared with a user-defined threshold v_t. If $\left\|\mathbf{v}_i^{diff}\right\| > v_t$, air is likely to be trapped at the position $\mathbf{x}_i$.

In order to relate the velocity of the inflow with the volume of the generated air, two further conditions must be met. First, the magnitude of the velocity must be greater than a predefined threshold $\|\mathbf{v}_i\| > v_{min}$, see Figure 6.3. Second, the number of air particle neighbors n_{air} of particle i should not be greater than $\left\|\mathbf{v}^{diff}\right\| / v_t$. Consequently, the number of generated particles grows with the velocity of the inflow.

In summary, a liquid particle i generates an air particle if $\left\|\mathbf{v}_i^{diff}\right\| > v_t$, $\|\mathbf{v}_i\| > v_{min}$ and $n_{air} < \left\|\mathbf{v}^{diff}\right\| / v_t$. The position and velocity of the air particle are chosen to be the same as for the liquid particle i.

Our model computes the density and pressure of the air and liquid phase separately. Therefore, an air particle can be generated at the position of a liquid particle without introducing high pressure forces causing unnatural acceleration. This is in contrast to the multiphase methods presented in [MSKG05, SP08] where high pressures are introduced when the distance of air and liquid particles is too small. Thus, in these models, determining an ap-

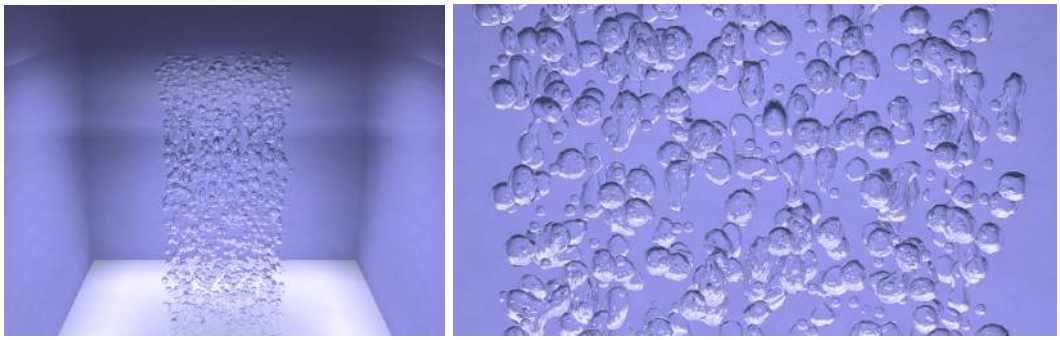

Figure 6.4: Rising bubbles in calm water. Bubbles are different in shape and size due to the cohesion force.

propriate position for a generated air particle without causing numerical instabilities is not straightforward.

Deletion. When an air particle reaches the surface it is given a *floating time* t_f, i.e,. time until the particle is deleted. In order to improve the realism, we vary t_f for each particle randomly using a uniform distribution. However, if two floating air particles merge, i.e., get neighbors, the minimum of their floating times is assigned to both particles. Thereby, bubbles consisting of more than one particle disperse at once.

6.3 Implementation

The proposed bubble model can be easily integrated into any existing SPH solver. In an SESPH implementation, the forces can be directly incorporated. However, in order to simulate water realistically, the compressibility should be set very low by using a relatively high speed of sound value. For the incompressible solvers, PCISPH [SP09] and IISPH, the proposed forces should be applied during the prediction step, i.e., before the pressure system is solved. Algorithm 5 provides an exemplary implementation using IISPH. The computational overhead imposed by the proposed model is negligible. Furthermore, the time step is not affected by the high density ratio of water and air. This is due to the velocity-based coupling which avoids the computation of pressure forces at the water-air interface.

Parameter setting. In all our experiments, we set the reference density of water to 1000 kg/m^3 and for air particles we set it to 1 kg/m^3. The particle mass is computed as $\rho_0/(0.5h)^3$, where $0.5h$ is the initial particle spacing, which is 0.02 in most of our scenes. The gravity is set to $(0, -9.81, 0)^T$. Velocities and accelerations are given in m/s and m/s^2, respectively.

For the bubbles, we empirically found the following setting to obtain good results: the buoyancy coefficients are set to $k_b = 14$ and $k_{max} = 6$, cohesion is set to $k_c = 12$ and the drag coefficient k_d is set to 8 for air particles and to 3 for water. We have varied the parameters for generating air bubbles in order to show their effect on the simulation (see Section 6.4.2).

Note that the coefficients introduced for the bubble model are independent of the resolution h. By increasing the parameters k_d, k_c and k_b, the effect of the corresponding force is amplified.

Algorithm 5 Air bubble model in IISPH. l indicates the iteration.

procedure PREDICT ADVECTION
  for all *particle* i **do**
    compute $\rho_i(t) = \sum_j m_j W_{ij}(t)$
    if $i \in$ air **then**
      compute $\mathbf{F}_i^{adv}(t) = \mathbf{F}_i^{cohesion} + \mathbf{F}_i^{buoyancy} + \mathbf{F}_i^{drag} + \mathbf{g}$
    else
      compute $\mathbf{F}_i^{adv}(t) = \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{drag} + \mathbf{g}$
    predict $\mathbf{v}_i^{adv} = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i^{adv}(t)}{m_i}$
    $\mathbf{d}_{ii} = \Delta t^2 \sum_j -\frac{m_j}{\rho_i^2} \nabla W_{ij}(t)$
  for all *particle* i **do**
    $\rho_i^{adv} = \rho_i(t) + \Delta t \sum_j m_j \left(\mathbf{v}_{ij}^{adv}\right) \nabla W_{ij}(t)$
    $p_i^0 = 0.5\ p_i(t - \Delta t)$
    compute a_{ii} (3.25)
procedure PRESSURE SOLVE
procedure INTEGRATION
  for all *particle* i **do**
    $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i^{adv} + \Delta t \mathbf{F}_i^p(t)/m_i$
    $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

6.4 Results

6.4.1 Bubble Flow

The basic capability of the presented bubble model is demonstrated by simulating rising bubbles in calm water (see Figure 6.4). In this example scene, air particles are randomly seeded at the bottom. The initial fluid velocity is zero. According to the proposed two-way coupling, the velocity field of the fluid is influenced by the air phase and vice versa. Thereby, small scale turbulences are generated which results in a natural zig-zag flow of the bubbles without adding random disturbance. As the example shows, the prominent effects of air-bubble flow can be successfully captured, e.g., merging, path instability, volume-dependent buoyancy.

In this scene, 2.4 million water particles and up to 10K air particles are simulated. The time step is set to 0.0015s which guarantees a compressibility of less than 1.5%.

6.4.2 Bubble Generation

A major contribution of the presented model is the generation of air bubbles. In contrast to existing work, air is generated in surface regions with high local velocity differences. According to the velocity-based coupling, bubbles can be generated on the fly without causing numerical instabilities. The generation of air bubbles is demonstrated in two example scenes.

In the first example, an inflow is simulated that is poured into a volume of water (see Figure 6.5). High velocity differences occur around the inflow. Accordingly, air particles are generated. Although, the bubble flow is mainly influenced by the turbulent velocity field of the liquid, the individual flow is quite different. Bubbles are varying in size and shape

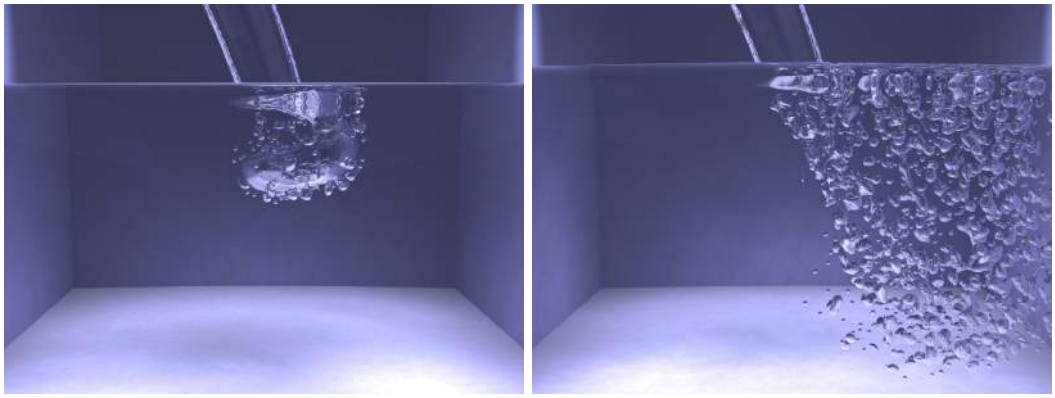

Figure 6.5: Trapped air. Air bubbles are generated on the fly in regions of high velocity differences. The bubble flow is significantly influenced by the liquid. Bubbles are merging and deforming. This simulation contains 1.4 million liquid particles and up to 6000 air particles.

according to the cohesion force. Due to the volume-dependent buoyancy, larger bubbles rise faster and more straight, while smaller bubbles are mainly influenced by the liquid. According to the proposed foam model, the air bubbles float realistically on the surface before they burst. For this example, we set the average floating time t_f to 0.7s. In this example, a liquid surface particle only generates an air particle if the velocity difference is larger than $v_t = 0.3$ and the magnitude of its velocity is larger than $v_{min} = 3$. Note that less air would be generated if v_t or v_{min} are set higher.

The second example demonstrates that the amount of generated air particles scales with the magnitude of the velocity difference. Therefore, three underwater inflows are simulated (see Figure 6.6). The velocities $v_n = (x_n, 0, 0)^T$ of the water inflows are set differently, where $x_1 = 5.5$, $x_2 = 7.0$ and $x_3 = 9.0$. v_t is set to 0.75 and v_{min} to 3.5. Thereby, the velocity differences around the inflow do vary. According to the proposed heuristic (see Section 6.4.2), most of the air particles are generated by the fastest inflow, while the slowest inflow generates significantly less bubbles. Furthermore, higher inflow velocities result in larger turbulences. Since these vorticities are mapped onto the bubble flow, the bubbles indicate the liquid flow. Note that without the animation of bubbles, the liquid inflow would not be visible in this example.

Again, spatially close air particles merge. Consequently, some air bubbles get quite large, particularly for the fast inflow. These large bubbles rise very fast due to the volume dependent buoyancy force. According to the drag force, they significantly influence the liquid as is clearly visible at the liquid surface. In both scenes, we set the time step to 0.002s.

6.5 Discussion

In this chapter, a new air bubble model for SPH has been presented. Numerical instabilities invoked by high density ratios at the air-liquid interface are avoided by coupling the two phases via the velocity field. Prominent phenomena of bubble flows are naturally captured, e.g., path instability. In contrast to existing models, the bubbles are simulated with SPH and not simply by solid spheres. According to the proposed cohesion force, effects like merging and deformation of bubbles can be successfully simulated. Furthermore, a velocity-based

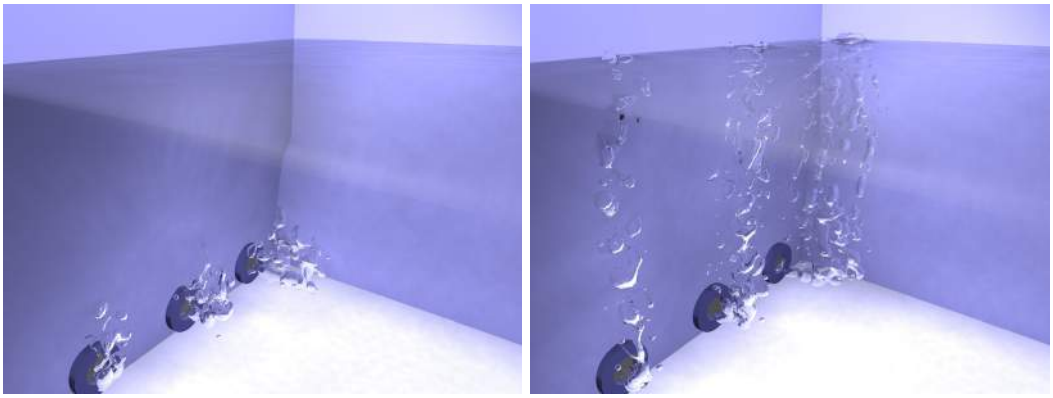

Figure 6.6: Underwater inflows. Water streams out of the three pipes with different velocities. The inflow in front has the lowest velocity and the inflow in the back the highest. Note that the amount of generated air scales with the velocity of the inflow. Larger air bubbles influence the liquid significantly. The simulation uses 650k water particles and up to 2k air particles.

heuristic that generates air bubbles for inflows has been proposed. Thereby, trapped air is animated efficiently, i.e., without explicitly simulating the air phase surrounding the liquid. By incorporating the bubble model into the PCISPH or IISPH method, large time steps can be used. This enables the generation of high-resolution animations where the bubbles are small in size in comparison to the liquid volume. This is demonstrated in the example scenes.

The proposed model does not address the interaction of air bubbles with rigid or deformable bodies. Effects like natural surface attraction of bubbles to solid surfaces can not be captured, but would certainly further improve the realism of the simulation.

Moreover, the performance can be further improved by using smaller support radii for air particles than for liquid particles. This is proposed for single-phase fluids in [DC96], [APKG07]. In scenarios where the influence of air bubbles onto water can be neglected, the air bubble model can be applied as a post-processing step. The generation of bubbles and their flow can be satisfactorily computed using the already simulated velocity field of the liquid. The next chapter discusses these ideas in detail and proposes an effective model for improving the realism of highly turbulent fluid simulations like rivers or waves.

CHAPTER 7

Secondary Simulation

The realistic animation of ocean scenes with breaking waves is challenging. Highly turbulent flows demand a high resolution to capture small-scale details like splashes, while for the major part of the fluid a lower resolution is sufficient. This can be parthandled by employing multi-scale methods, e.g., for Eulerian approaches [LGF04, KFCO06, IGLF06, CM11] and for Lagrangian approaches [APKG07, SG11, RWT11, OK12]. Furthermore, for realistic animations, the interplay of fluid and air has to be modeled. This interplay causes the formation of diffuse material, perceived as foam, spray and tiny air bubbles which is addressed in this chapter.

In order to simulate fluid-air mixtures, multi-phase approaches which simulate the air phase explicitly have been proposed [TFK$^+$03]. However, the high density contrast of water and air is numerically challenging and restricts the time step. Therefore, some methods propose to model the air phase implicitly. In these approaches, air-water mixtures are determined based on hypothetical models. Implicit modeling is convenient to capture even small-scale phenomena. In the context of air bubble animations, this has been demonstrated for grid-based [HLYK08, PAKF13] and particle-based solvers [MSKG05, GH04, IBAT11], whereas implicit mist models and spray models [TRS06, LTKF08, CM11] have been coupled only to grid-based fluid solvers. However, no method exists that simulates spray, foam and air-bubbles in an unified way. In this chapter, such a method is proposed for Lagrangian fluids.

As the influence of diffuse material onto the water phase can be neglected for large-scale simulations, the proposed model is realized as a secondary simulation on pre-computed fluid data. The proposed method does not employ a set of techniques for different types of diffuse material, but solely relies on particles. Therefore, physically-motivated criteria are formulated for determining regions where water mixes with air. In these regions, the amount of air concentration determines the number of generated and dissolved particles that represent diffuse material. It should be noted that appropriate criteria for the generation of diffuse particles are essential in order to obtain realistic flow patterns over time. The advection of diffuse particles is not based on interparticle forces, but mainly determined by the velocity field of the water phase. Thus, expensive neighborhood computations are avoided. Furthermore, large time steps up to the frame rate of the animation can be handled.

The presented model for diffuse material requires a minimal set of input parameters which can be provided by any Lagrangian fluid simulation. Due to its computational efficiency and the intuitive parameter setting, the proposed model is an attractive tool to improve the visual realism of existing large-scale particle simulations. A first example is given in Figure 7.1.

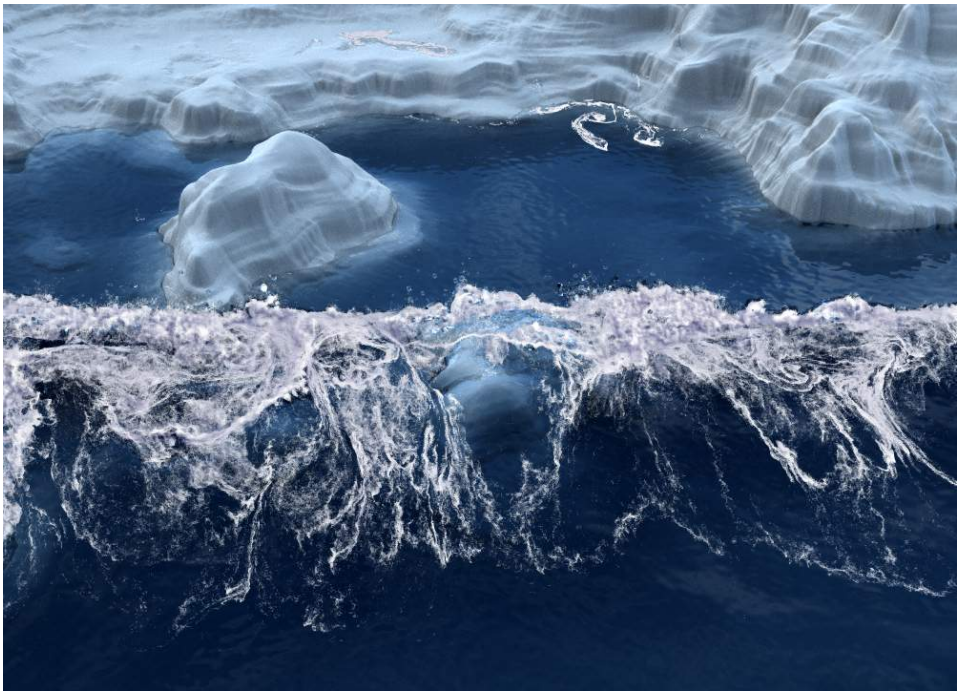

Figure 7.1: Water waves simulated with 16 million SPH particles of volume radius 1.8cm. Up to 25 million air bubbles, spray and foam particles are added in a secondary simulation to improve the visual realism. Secondary particles represent a volume radius of 2mm.

7.1 Related Work

Foam and splash effects are important features of realistic fluid simulations [TMFSG07, CM10], movies [FAZ+00, GLR+06, BSK+07] and commercial software like Houdini (Side Effects Software Inc.) and RealFlow (Next Limit) [Nex11]. These works, however, only briefly describe the generation, advection and dissolution of diffuse material. Furthermore, they combine different techniques, e.g., particles and textures.

In computer graphics, only a few models have been presented for the unified simulation of diffuse materials. In [TFK+03], a model for simulating splashes and foam is proposed which couples the particle-based diffuse material with a grid-based fluid solver. Splash and foam particles are generated when the surface curvature of the fluid exceeds a threshold. State change rules are employed to advect and dissolve splashes and foam. The proposed model shares the motivation of [TFK+03]. Alternative criteria are presented which are designed for Lagrangian fluids. Furthermore, in contrast to [TFK+03], the criteria are effective in creating diffuse material not only at the crest of a wave, but also in capturing the entrainment of air.

A common strategy for grid-based solvers is to generate secondary particles in under-resolved regions [FF01, GH04]. In [KCC+06], massless marker particles that are escaping from the main body of water are transformed into particles to represent subcell-level features like splashes and droplets. This idea is adopted in [LTKF08] to simulate splash and spray. In this method, dense fluid regions are solved on a grid using the combination of FLIP and PIC described in [ZB05], while the interface is modeled by the particle level set method [FF01].

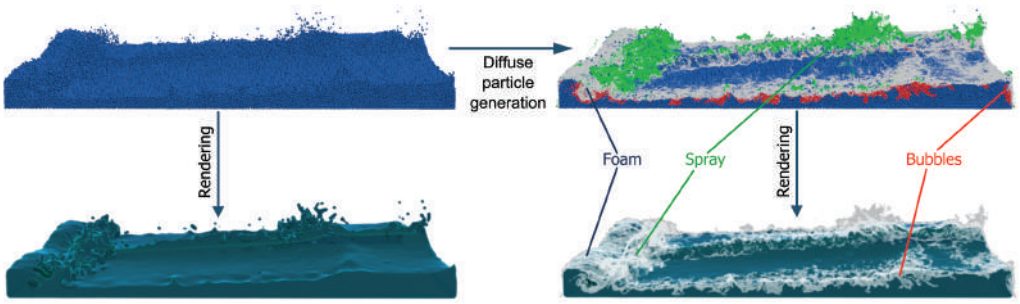

Figure 7.2: The proposed model is applied as a post-process step for single-phase particle-based fluids (top left). Air-water mixtures like spray, foam and bubbles are generated, advected and dissolved (top right). For rendering, the fluid's surface is triangulated neglecting isolated particles. A volumetric render technique is employed for diffuse material (bottom right).

Instead of removing particles that cross the interface, they are further simulated as diffuse material using the SPH method. Air-water mixtures are naturally handled without employing explicit criteria for the generation of diffuse material. Impressive results could be achieved by the two-way coupling of dense and diffuse materials. However, this approach is restricted to the level set method, whereas our model can be applied to any particle-based fluid. Moreover, [LTKF08] primarily focuses on capturing small-scale details which would be lost otherwise. In contrast, our model considers the advection and dissolution of different types of diffuse materials. Furthermore, forces between diffuse particles are not computed which avoids expensive neighborhood computations.

Various approaches exclusively employ geometric criteria like curvature to detect air-water mixtures, e.g., [GSLF05, MSKG05]. However, as discussed in [MMS09], fluid dynamics also plays an important role. Mihalef et al. determine the generation of small-scale splashes according to the relation of the kinetic energy and the surface energy which is referred to as the Weber number [Sir99]. In this approach, the large-scale flow is simulated with the marker level set method [MMS07], while small-scale details are modeled with a particle system. Splashes and air bubbles could be realistically captured with this model, however, the simulation and rendering of floating foam is not discussed.

Bagar et al. [BSW10] use the Weber number to classify fluid regions into water and foam for real-time rendering of SPH fluids. In this method, fluid particles are rendered as foam particles when the Weber number exceeds a threshold. Thereby, the realism of real-time simulations is significantly improved. The simulation, however, does not distinguish between foam and fluid particles and does not generate additional particles. Thus, the level of detail is restricted to the resolution of the underlying fluid. In contrast, our model generates diffuse particles with an initial momentum based on the kinetic energy of the fluid. Furthermore, the level of detail of the diffuse material is scalable.

7.2 Spray, Foam and Bubbles

In the proposed model, the air phase is simulated implicitly in order to avoid numerical instabilities, small time steps and expensive computations. In each simulation step, a potential

to mix with air is computed per fluid particle. If there is some potential, a fluid particle generates diffuse particles. Based on their location, particles are classified as foam, spray or bubbles (see Figure 7.2). The type determines how particles are advected. In Section 7.2.1, we describe how the potential for diffuse material is computed. Furthermore, we explain how the position and velocity of generated diffuse particles are initialized. Subsequently, we state how diffuse particles are advected and dissolved in Section 7.2.2. An exemplary rendering technique for diffuse material is provided in Section 7.2.3.

7.2.1 Formation of Secondary Particles

In nature, diffuse material develops when the surface tension of water molecules is reduced and water mixes with air. E.g., if the crest of a wave gets unstable, water becomes aerated, resulting in turbulent whitewater which either spills down as foam or sprays away from the wave as mist. The activity of mixing might be caused by high wind and wave speeds, but also by trapped air due to high velocity differences at the fluid surface. In both cases, the amount of generated diffuse material increases with the kinetic energy of the fluid.

In our model, the potential of a fluid particle to mix with air is determined by its potential to trap air, its likelihood to be at the crest of a wave and its kinetic energy. A fluid particle generates diffuse particles if the combined potential is larger than zero. In order to scale the amount of newly generated diffuse material and, thus, the level of detail, the user can provide minimum τ^{min} and maximum τ^{max} thresholds for each criterion. According to these thresholds, the potential I for the generation of diffuse material is mapped to the range between zero and one using a clamping function Φ which is defined as

$$\Phi(I, \tau^{min}, \tau^{max}) = \frac{\min(I, \tau^{max}) - \min(I, \tau^{min})}{\tau^{max} - \tau^{min}}. \tag{7.1}$$

We now describe the proposed potential followed by an explanation of how the positions and velocities of generated particles are sampled.

Trapped air. Air is trapped by impacts, e.g., when the lip of a wave hits shallow water. In this case, air is dragged under water. Furthermore, high turbulences cause aerated splashes. In order to determine those regions, the curl operator $\nabla \times \mathbf{v}$ might be a good choice. However, while the curl is large in turbulent regions, it might be small for impacts. Therefore, we propose to use relative velocities to determine regions where air is potentially trapped since these are large for impacts and vortices. We further assume that the amount of trapped air is larger if the fluid particles move towards each other which can be measured by $(1 - \hat{\mathbf{v}}_{ij} \cdot \hat{\mathbf{x}}_{ij})$, with $\hat{\mathbf{v}}_{ij} = \frac{\mathbf{v}_i - \mathbf{v}_j}{\|\mathbf{v}_i - \mathbf{v}_j\|}$ denoting the normalized relative velocity between two particles and $\hat{\mathbf{x}}_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$ their normalized distance vector. This term is zero for particles that move away from each other while it is two for particles that move towards each other. The scaled velocity difference

$$v_i^{diff} = \sum_j \|\mathbf{v}_{ij}\| \left(1 - \hat{\mathbf{v}}_{ij} \cdot \hat{\mathbf{x}}_{ij}\right) K(\mathbf{x}_{ij}, h) \tag{7.2}$$

is then used to compute the trapped-air potential as $I_{ta} = \Phi(v_i^{diff}, \tau_{ta}^{min}, \tau_{ta}^{max})$. $K(\mathbf{x}_{ij}, h)$ is a radially symmetric weighting function defined as

$$K(\mathbf{x}_{ij}, h) = \begin{cases} 1 - \|\mathbf{x}_{ij}\| / h & \|\mathbf{x}_{ij}\| \leq h \\ 0 & \text{otherwise} \end{cases}, \tag{7.3}$$

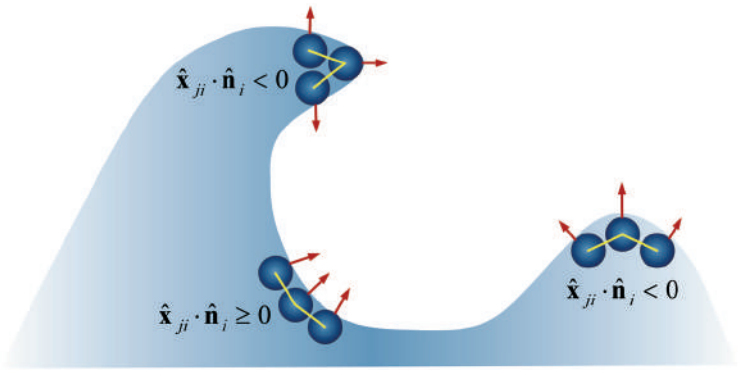

Figure 7.3: Convex and concave regions are determined using the angle between the surface normal of a fluid particle and the relative position vectors to its neighbors. Diffuse particles are created in convex regions only if the velocity of the fluid is in normal direction.

where h is the influence radius of the fluid simulation. Compared to commonly employed kernels, e.g., cubic spline [Mon92], we found this weighting function to give better estimates near free surfaces where the neighborhood of fluid particles is not sufficiently sampled.

Wave crest. At the crest of a wave, whitewater is created either by strong winds or if the wave gets unstable, i.e., if the wave base can no longer support its top. In order to identify these areas, we assume that the surface curvature is high and the surface is locally convex. For a set of points, the surface curvature κ can be approximated with

$$\kappa_i = \sum_j \kappa_{ij} = \sum_j (1 - \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j) K(\mathbf{x}_{ij}, h), \tag{7.4}$$

where $\hat{\mathbf{n}}$ is a normalized surface normal. In order to distinguish convex from concave regions, the angles between $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{x}}_{ji}$ are considered (see Figure 7.3). Accordingly, wave crests are identified using

$$\tilde{\kappa}_i = \sum_j \tilde{\kappa}_{ij}, \tag{7.5}$$

with

$$\tilde{\kappa}_{ij} = \begin{cases} 0 & \hat{\mathbf{x}}_{ji} \cdot \hat{\mathbf{n}}_i \geq 0 \\ \kappa_{ij} & \hat{\mathbf{x}}_{ji} \cdot \hat{\mathbf{n}}_i < 0 \end{cases}. \tag{7.6}$$

Experiments show that (7.5) is a good estimate to identify wave crests. However, (7.5) would also identify all edges of, e.g., a cube of water as wave crests. Therefore, we propose to additionally check if the fluid particle moves in normal direction using

$$\delta_i^{vn} = \begin{cases} 0 & \hat{\mathbf{v}}_i \cdot \hat{\mathbf{n}}_i < 0.6 \\ 1 & \hat{\mathbf{v}}_i \cdot \hat{\mathbf{n}}_i \geq 0.6 \end{cases}. \tag{7.7}$$

Finally, the likelihood of a particle to be at the crest of a wave is computed as $I_{wc} = \Phi(\tilde{\kappa_i} \cdot \delta_i^{vn}, \tau_{wc}^{min}, \tau_{wc}^{max})$.

Energy. In fluid dynamics, the Weber number is a useful quantity for analyzing the air entrainment and the formation of droplets. However, its exact computation requires to correctly model the change in surface tension, for example caused by strong winds or the influence of solute concentrations. Since this is challenging to model, [MMS09] and [BSW10]

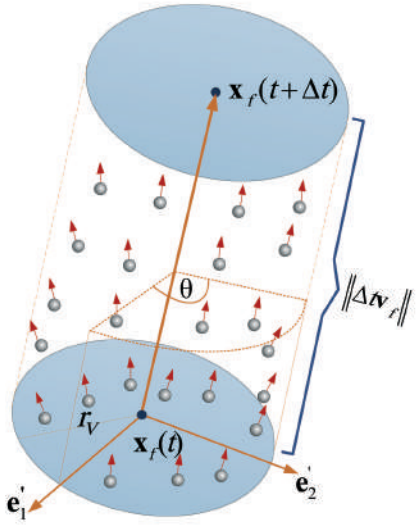

Figure 7.4: Diffuse particles are generated by a fluid particle f. They are uniformly distributed in a cylinder spanned by the volume radius r_V, the current position $\mathbf{x}_f(t)$ and the position $\mathbf{x}_f(t+\Delta t)$ of the fluid particle. The velocities are initialized according to the positions and the fluid velocity $\mathbf{v}_f$.

assume a constant surface tension. As changes in the surface tension are not considered, only the kinetic energy $E_{k,i} = 0.5 m_i \mathbf{v}_i^2$ is used as a measurement for air entrainment. Therefore, we relate the amount of diffuse material generated by a fluid particle to its kinetic energy. Accordingly, the potential to generate diffuse particles due to kinetic energy is computed as $I_k = \Phi(E_{k,i}, \tau_k^{min}, \tau_k^{max})$ with user-defined τ_k^{min} and τ_k^{max}.

Sampling. The proposed criteria are finally composed in order to compute the amount of generated diffuse material for each fluid particle. Here, we assume that air either mixes with water at the crest of a wave or when air is trapped. In both cases, the amount of mixed air scales with the kinetic energy. Due to the mapping function Φ, all three potentials are in the range between zero and one. The user controls the amount of generated diffuse material per particle and time step by providing a maximum number of samples that should be generated on wave crests k_{wc} and for trapped air k_{ta} per second. Consequently, the number of diffuse particles generated by a fluid particle is computed as

$$n_d = I_k \left(k_{ta} I_{ta} + k_{wc} I_{wc}\right) \Delta t, \tag{7.8}$$

where Δt denotes the time step.

We propose to sample the generated particles in a cylinder spanned by the volume radius r_V of a fluid particle and its velocity $\mathbf{v}_f$. Therefore, we compute a reference plane orthogonal to $\mathbf{v}_f$, spanned by $\mathbf{e}_1'$ and $\mathbf{e}_2'$ with the fluid particle position $\mathbf{x}_f$ as the origin. The position of a diffuse particle $\mathbf{x}_d$ is computed according to three uniformly distributed random variables $X_r, X_\theta, X_h \in [0..1]$. They determine the distance to the cylinder axis $r = r_V \sqrt{X_r}$, the azimuth $\theta = X_\theta 2\pi$ and the distance $h = X_h \cdot \|\Delta t \mathbf{v}_f\|$ from the reference plane (see Figure 7.4). Accordingly, $\mathbf{x}_d$ is computed as $\mathbf{x}_d = \mathbf{x} + r\cos\theta \mathbf{e}_1' + r\sin\theta \mathbf{e}_2' + h\hat{\mathbf{v}}_f$. Thereby, particles are uniformly sampled in the cylinder.

The proposed model does not compute internal forces for diffuse particles. They are mainly

advected by the velocity field of the fluid as explained in the subsequent section. In order to simulate splashes into different directions, the velocities of newly created diffuse particles are initialized as $\mathbf{v}_d = r\cos\theta\mathbf{e}_1' + r\sin\theta\mathbf{e}_2' + \mathbf{v}_f$ where $\mathbf{e}_1'$ and $\mathbf{e}_2'$ act as velocities.

It should be noted that an appropriate sampling is important to obtain a volumetric appearance of diffuse material and also to avoid regular patterns.

7.2.2 Advection and Dissolution

As long as diffuse material is not influenced by water, its motion can be assumed to be ballistic, whereas under water, diffuse material is highly influenced by the water phase. Water exerts high pressures on the buoyant air bubbles which are under strong velocity diffusion. When rising bubbles reach the surface, they form floating foam which is transported by the liquid.

In accordance with these observations, we do not compute internal forces for diffuse particles, but determine their motion using the fluid velocities and external forces. Since the influence of the fluid depends on the location, we classify the diffuse material into spray, foam and bubbles (see Figure 7.2). A diffuse particle d is classified as spray, foam or air bubble according to its fluid particle density $\tilde{\rho} = \sum_f W(\mathbf{x}_d - \mathbf{x}_f, h)$, where f denotes fluid particles and W should be a normalized, symmetric kernel, e.g., the cubic spline kernel [Mon92]. At the surface, a diffuse particle has less fluid neighbors than inside the fluid volume. Consequently, inside the fluid volume, the particle density is always higher than on the surface, at least for weakly or incompressible fluids. Therefore, surface particles can be either determined by the gradient of the density field [MCG03] or simply by the number of neighbors. In all our experiments, diffuse particles with less than 6 fluid neighbors are considered as spray particles. Particles with more than 20 fluid neighbors are classified as air bubbles. In all other cases, particles are considered to be foam.

The motion of spray is computed solely using the momentum, external forces $\mathbf{F}_{ext}$ and gravity $\mathbf{g}$. Using the Euler-Cromer method, the velocity of a spray particle is updated as $\mathbf{v}_{spray}(t+\Delta t) = \mathbf{v}_{spray}(t) + \Delta t(\frac{\mathbf{F}_{ext}(t)}{m} + \mathbf{g})$, where m is the mass of the spray particle. The position is then updated as

$$\mathbf{x}_{spray}(t+\Delta t) = \mathbf{x}_{spray}(t) + \Delta t\mathbf{v}_{spray}(t+\Delta t).$$

In contrast, foam and bubble particles are highly influenced by the fluid. Foam is purely advected according to the averaged local fluid velocity at the position of the diffuse particle d which is computed as

$$\tilde{\mathbf{v}}_f(\mathbf{x}_d, t+\Delta t) = \frac{\sum_f \mathbf{v}_f(t+\Delta t)W(\mathbf{x}_d(t) - \mathbf{x}_f(t), h)}{\sum_f W(\mathbf{x}_d(t) - \mathbf{x}_f(t), h)},$$

where $\mathbf{v}_f(t+\Delta t) = \frac{\mathbf{x}_f(t+\Delta t) - \mathbf{x}_f(t)}{\Delta t}$. In order to constrain foam particles to the fluid surface, we only update their positions as

$$\mathbf{x}_{foam}(t+\Delta t) = \mathbf{x}_{foam}(t) + \Delta t\tilde{\mathbf{v}}_f(\mathbf{x}_d, t+\Delta t),$$

but not their velocities.

Due to the high density contrast of water and air, the motion of air bubbles is additionally governed by buoyancy which counteracts gravity. Therefore, the velocity of an air bubble is

computed as

$$\mathbf{v}_{bub}(t+\Delta t) = \mathbf{v}_{bub}(t) + \Delta t \left(-k_b \mathbf{g} + k_d \frac{\tilde{\mathbf{v}}_f(\mathbf{x}_d, t+\Delta t) - \mathbf{v}_{bub}(t)}{\Delta t} \right),$$

where k_b and k_d are user-defined constants which control the buoyancy and drag effects. It should be noted that if k_d is chosen as 1, air bubbles are immediately dragged into the flow direction of the fluid. The position of an air bubble is then updated as

$$\mathbf{x}_{bub}(t+\Delta t) = \mathbf{x}_{bub}(t) + \Delta t \mathbf{v}_{bub}.$$

Dissolution. When diffuse particles are created, their lifetime is initialized with a predetermined value. In each simulation step, the time step is subtracted from the lifetime of foam particles, whereas the lifetime of air bubbles and spray particles is not reduced. Foam particles are finally deleted when the lifetime is smaller or equal to zero. In nature, large clusters of foam are more stable than smaller ones. In order to capture this effect without determining the foam area, we set the lifetime in relation to the generation potentials. Thus, the lifetime is not constant, but in the range between a user-defined minimum and maximum value.

7.2.3 Implementation and Rendering

In the proposed model, interparticle forces for the diffuse material are not computed. Therefore, the diffuse neighbors of a diffuse particle are not required. However, for determining the formation potentials and for the integration of particles, neighborhood relations between fluid particles, as well as for diffuse and fluid particles need to be computed. This can be efficiently computed with, e.g., compact hashing or Z-index sort [IABT11] as presented in Chapter 4.

The sequences are rendered with mental ray 3.9 for which we have implemented a volume shader based on ray casting which accounts for absorption and emission of radiance, but neglects scattering effects. Accordingly, for each pixel of the final image, an eye ray is cast through the diffuse volume which is bounded by an axis aligned bounding box with entry point $\mathbf{x}_s$ and exit point $\mathbf{x}_e$ for a considered ray. The ray is sampled using equally spaced intervals $\Delta\mathbf{x}$. For each sample point $\mathbf{x}$ on a ray with direction ω, the local volume density $\rho(\mathbf{x})$ is computed. Based on the density at each sample point $\mathbf{x}$, the transparency T along ω is computed as

$$T(\mathbf{x}, \omega) = \prod_{i=0}^{s} e^{-\rho(\mathbf{x}_s + i\Delta\mathbf{x})\tau\Delta\mathbf{x}}, \tag{7.9}$$

where $s = \left\lfloor \frac{\mathbf{x}-\mathbf{x}_s}{\Delta\mathbf{x}} \right\rfloor$ and τ is a user-defined scaling factor. T is in the range between zero and one, where zero means fully opaque and one means fully transparent. Let C_d be the user-defined color of the diffuse material and C_b the background color. The final color C_p for the corresponding pixel is then determined as $C_p = (1 - T(\mathbf{x}_e, \omega))C_d + T(\mathbf{x}_e, \omega)C_b$.

The volume density $\rho(\mathbf{x})$ is computed using constant thresholding. It is zero if there is no diffuse particle in the volume radius r_d around $\mathbf{x}$ and otherwise it is one. In order to make spray look sharper than foam, we use different volume radii for spray and foam. Please note that alternative weighting functions could be employed, e.g., a Gaussian function. However, we found constant thresholding to give the best results.

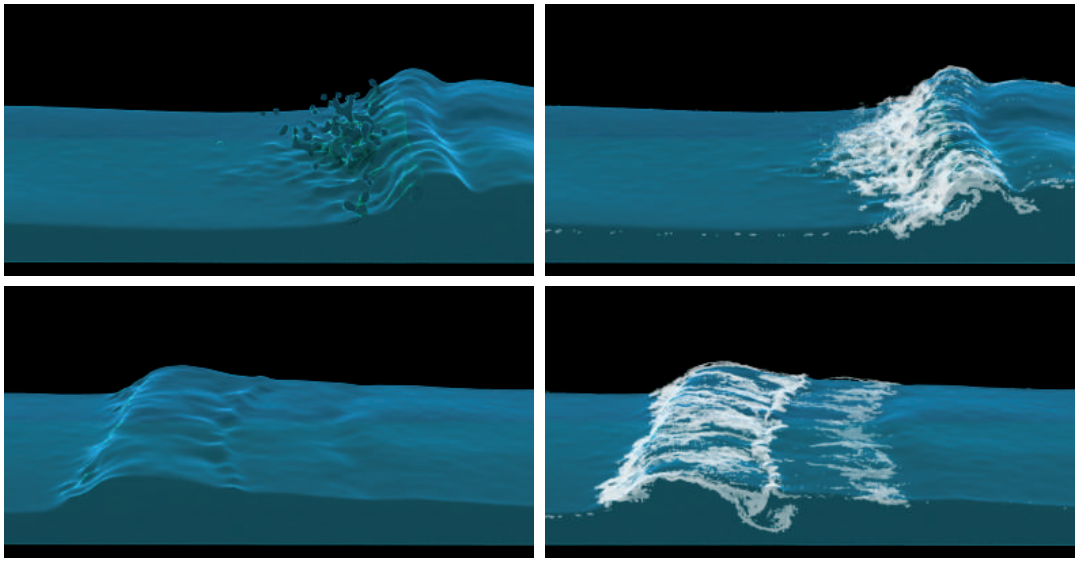

Figure 7.5: Post-processing of a single-phase wave simulation. Water-air mixtures are generated at the crest of the wave and in the impact zone of the wave. The realism of the coarse fluid simulation is significantly improved by adding multiphase effects with small-scale features.

Scene	# particles fluid	# particles diffuse	computation times fluid	computation times diffuse	frames
Wave	220k	2.7m	53 min	28 min	1000
Tower	3.5m	7m	995 min	81 min	700
Ship	2m	2m	185 min	25 min	400
Lighthouse	640k	3.3m 15m	420 min	35 min 121 min	1000

Table 7.1: Timings and particle counts for the presented showcases.

7.3 Results

We discuss four scenarios which show different aspects of our approach and we evaluate the performance. The timings presented in Table 7.1 are measured on a 3.33 GHz Six-Core Intel machine. The underlying incompressible SPH simulations have been computed with PCISPH. In all scenarios, diffuse material is simulated with respect to the frame rate of the fluid simulation which is 50 frames per second, resulting in a time step of 0.02 seconds. The volume radius of diffuse particles is ten times smaller than the radius of fluid particles. For all scenarios, we found the following thresholds to give good results: $\tau_{wc}^{min} = 2$, $\tau_{wc}^{max} = 8$, $\tau_{ta}^{min} = 5$, $\tau_{ta}^{max} = 20$, $\tau_{k}^{min} = 5$ and $\tau_{k}^{max} = 50$.

7.3.1 Generation of Diffuse Material

The wave scene (see Figure 7.5) is a basic test for the generation criteria. It features a cuboid of low-viscous water that is dropped into a box. Due to the initialization, regions with high

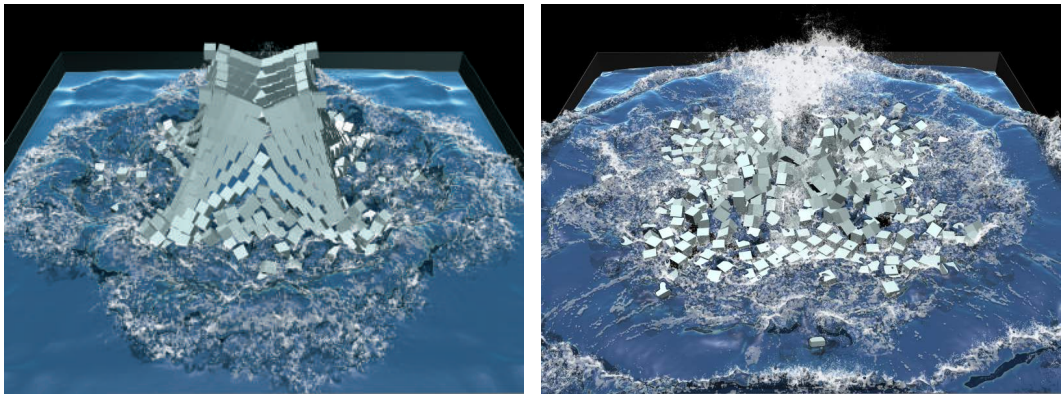

Figure 7.6: A tower with 1000 cubes collapses. Diffuse material is generated at the crest of waves, for splashes and where air is trapped.

surface-curvature and velocity differences occur. Additionally, waves are generated by a moving plane. Due to the low resolution of the fluid, small-scale details are not captured and isolated fluid splashes are too coarse.

Diffuse material is plausibly generated at wave crests and in regions with high velocity differences, while it is not generated at sharp features at the boundary. This scene captures many prominent effects like the formation of foam at the lip of the wave and the creation of whitewater in the so-called surf zone. The amount of generated diffuse material depends on the kinetic energy of the wave. Furthermore, the foam patterns and the air bubbles emphasize the dynamics of the fluid. The criteria are also working in more complex settings as demonstrated by the tower scene (see Figure 7.6).

7.3.2 Air Entrainment by Moving Objects

Air entrainment caused by moving objects is a prominent effect that can be observed in nature. In order to demonstrate that our model is able to capture this effect, we employ an animated ship within a fluid simulation. Spray and foam are plausibly generated at the bow and the stern of the ship, resulting in realistic foam patterns (see Figure 7.7). The visual appearance of the base simulation is significantly improved as also demonstrated in the accompanying video.

7.3.3 Scaling of Quality and Performance

The animator can control the level of detail by defining the maximum amount of generated diffuse material on wave crests and in regions of high velocity differences (see Section 7.2.1). As a showcase for the scaling of quality and performance, a large-scale domain with a lighthouse is flooded. Large velocity differences and turbulences occur around the lighthouse, while larger waves form at the wall in the back.

In the left column of Figure 7.8, an air-trapping fluid particle generated up to 30 diffuse particles per frame, while for wave crests, the maximum number of samples has been set to 50. Thereby, up to 3.3 million diffuse particles have been simulated per frame.

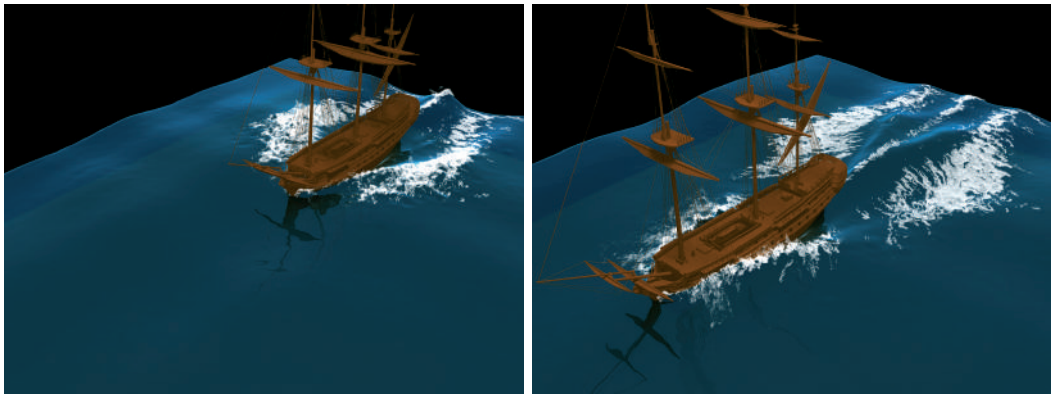

Figure 7.7: Ship scene. Air entrainment of moving objects can be realistically captured as the foam pattern illustrates.

In the right column of Figure 7.8, the number of generated samples has been increased by a factor of 4.5 for both, trapped air and wave crests. Thereby, up to 15 million diffuse particles have been simulated per frame. As also illustrated in the accompanying video, the additional diffuse particles significantly improve the level of detail. This is due to the proposed sampling which distributes positions and velocities of diffuse particles uniformly in a cylinder spanned by the current and next frame position of the generating fluid particle.

For each scenario, the performance and the level of detail scale with the user-defined maximum thresholds for the number of generated samples. Since increasing these thresholds does not influence where diffuse material is generated, the animator can first adjust the minimum and maximum thresholds of the generation potential and then increase the resolution.

7.4 Discussion

In this chapter, a unified method has been proposed which adds diffuse material to single-phase particle fluids in a post-processing step. Diffuse particles like foam, spray and air bubbles are either generated at the crest of a wave or in regions where air is likely to be trapped. The amount of generated diffuse particles scales with the estimated air concentration and the kinetic energy. The proposed criteria together with the initialization of position and momentum, and the velocity-based one-way coupling are appropriate to realistically simulate the dynamics of diffuse material, resulting in visually plausible, highly detailed flow patterns. Expensive neighborhood computations are avoided and large time steps can be handled. The presented showcases demonstrate that the realism of fluid simulations is significantly improved especially for low resolution fluids.

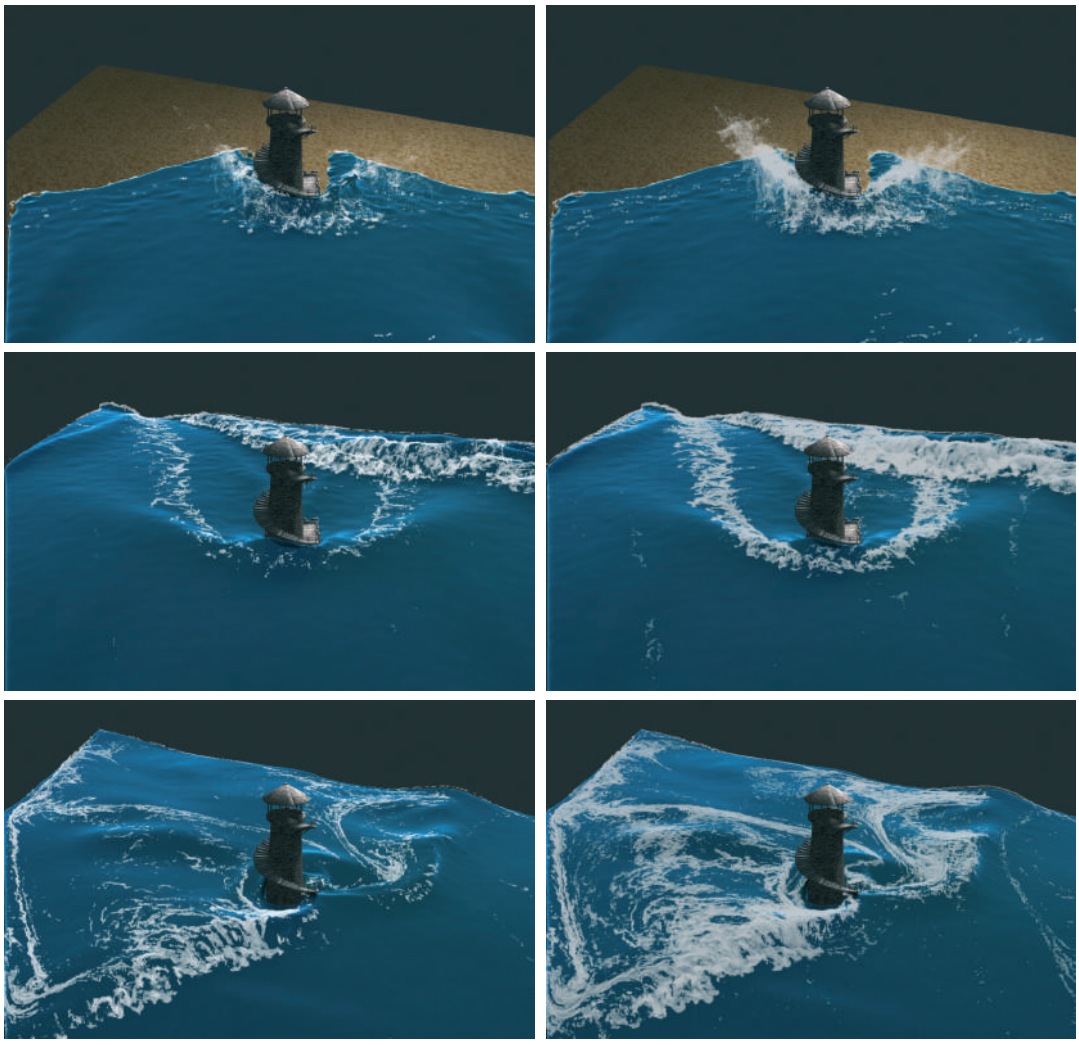

Figure 7.8: Lighthouse scene. Foam and spray effects are added in a post-process using up to 3.3 million (top) and up to 15 million (bottom) diffuse particles. The level of detail can be controlled without changing the overall behavior.

CHAPTER 8

Multiple Resolutions

The accuracy and realism of physically-based simulation techniques scale with the resolution. However, higher resolutions also induce longer computation times. This is not only due to an increase in the number of particles, but also caused by the fact that each particle represents a smaller volume which is a limiting factor for the time step. In order to enforce numerical convergence in SPH, each particle should not move more than half the support radius per time step. This is known as the CFL condition which in technical terms states that the numerical domain of dependence always includes the analytical domain of dependence. This means in practice that doubling the resolution implies halving the time step. However, for convincing simulations of materials like sand a relatively high resolution is required. As modeling the physical behavior by simulating each physical grain imposes prohibitively high computational costs, an alternative method is required. This chapter addresses this issue by proposing a multi-resolution simulation framework for the simulation of sand.

Granular materials, such as sand, rice or coffee beans are conglomerations of discrete solid elements which show unique physical behavior. They settle in stable piles and act like a solid if the average energy is low. When freely flowing, they have similar characteristics as ordinary Newtonian fluids, but unlike fluids, granular material dissipates energy quickly. The complex dynamics arise from the interplay of contact forces between the elements.

In order to capture this behavior, various simulation methods have been developed in the engineering field, e.g., [Bic04, JLL04], and also in computer animation, e.g. [LM93, SOH99, HBK09, NGL10]. In contrast to computational mechanics, the main focus in animation is set on efficient techniques that achieve visually plausible results. In order to realize an efficient implementation, simplifying assumptions and a coarse discretization are employed. While this avoids simulating each physical grain, it poses a new major challenge, namely how to achieve rich visual detail.

This chapter proposes a new Lagrangian simulation framework for capturing granular dynamics with high visual detail at low computational costs. Appropriate mechanical behavior is modeled by computing frictional and pressure forces on a coarse scale. High visual detail is obtained in a post-process, where a spatially fine-scaled set of particles is coupled to the base simulation. Our model aligns high-resolution particles with the surface of the base simulation without perceivable clumping. Since external forces are also acting on the fine resolution, they can depart freely from the base simulation. The presented upsampling method increases the visual quality dramatically without uncovering the base simulation, even for large upscaling factors and scenarios with dynamic objects, see Figure 8.1.

Figure 8.1: The proposed framework simulates mechanical behavior at a coarse scale (left), 38K particles. The base simulation is significantly refined with a secondary simulation (right), 19.4 million particles.

8.1 Related Work

In computer graphics, granular media is simulated using either discrete [Bic04, LHM95] or continuum methods [ZB05]. In discrete models, the material is discretized into a distinct set of elements and the behavior is captured by directly simulating the interaction between these elements. In [BYM05], grains are modeled by rigid compounds of spheres. Due to the non-spherical shape of the compound structures, stable sand piles can be simulated. By generalizing this approach to rigid bodies, compelling two-way coupling between granular material and solid objects is achieved. Discrete models require a large number of particles to model a fine-grained material. This imposes high computational costs for solving contact dynamics and limits the time step for highly dynamic simulations.

In continuum methods, the grain size is decoupled from the resolution of the simulation. In such models, internal forces are typically computed on a coarse grid. The resulting velocity field is then used to advect a set of fine-scale particles. This was first demonstrated by Zhu and Bridson [ZB05], who simulated sand as an incompressible fluid using the FLIP method. In order to simulate stable piles, this method classifies the sand domain into regions which are either rigidly moving or flowing. Later, Lenaerts and Dutré [LD09] incorporated this concept to simulate granular material with the SPH method [MCG03]. However, the incompressibility assumption results in undesired cohesive behavior which prevents plausible animations of freely dispersing material. Narain et al. [NGL10] addressed this problem by employing an unilateral incompressibilty constraint, i.e., negative flow divergence is not counteracted by pressure. In the sense of FLIP, they solve the internal forces on an Eulerian grid. The resulting velocities are then used to advect the particles, representing the material. Recently, Alduan and Otaduy [AO11] adapted unilateral incompressibility to the PCISPH method.

The purely Lagrangian framework proposed in [AO11] does not suffer from grid artifacts which is a major benefit compared to the Eulerian framework described in [NGL10]. However, it also introduces new challenges. Following the SPH concept, the pressure is computed based on density values and not according to the divergence of the velocities. This makes the approach very sensitive to sudden increases in the density. Oscillations in the density field particularly occur at interfaces with dynamic solid objects due to particle deficiency. In the context of fluid simulations, this has been addressed in [IAGT10, AIS+12] by treating the boundary as an interface to the fluid simulation. Thereby, spatial and temporal

discontinuities of physical properties are avoided, resulting in a smoother and a more robust simulation compared to commonly employed distance-based penalty methods, e.g., [MST+04, BIT09]. Furthermore, in [AO11], internal forces and advection are computed using the same discretization scale. Thus, in order to compute fine-grained material, significantly higher computational costs are imposed compared to [NGL10].

In this chapter, we address these challenges by extending the pure Lagrangian, continuum framework [AO11] in two ways. First, we show that the physically-based rigid-fluid coupling presented in [AIS+12] can be easily adapted to handle smooth interactions with granular material. This eliminates oscillations in the pressure field. As a consequence, less iterations are required when solving for the internal forces even at larger time steps. The second extension enables the simulation of fine-grained material at low computational costs by refining the simulation in a post-process.

The general idea of refining the coarse simulation for rendering is not new. Most authors propose to sample the simulation particles with a finer set of pseudo-random particles [LD09, NGL10]. High-resolution (HR) particles are fixed to the base particles in order to avoid temporal flickering. As HR particles can not disperse freely, such an approach results in clumping artifacts, perceived as staircase or spherical patterns and a distorted distribution of the material. Narain et al. [NGL10] addressed this problem by inserting additional anisotropic particles in regions where the material diverges. This step is already performed during simulation. In contrast, [ABC+07, ATO09] employ a spatial decomposition for the computation of internal and external forces. In sparse regions, HR particles are not passively advected with the base simulation, but respond to external forces. Thereby, clumping artifacts are significantly reduced.

Our refinement method is inspired by the decomposition idea, but departs significantly from previous work. In [ATO09], HR particles are either advected according to external forces or along the base flow. In contrast, our model does not rely on two complementary cases, but blends external forces with the interpolated base velocity. This yields more natural looking results, particularly for large upscaling factors. Furthermore, as we take velocities of boundaries into account, HR particles interact smoothly with complex moving objects.

The rest of the chapter is structured as follows: Section 8.2 presents a method for simulating sand with SPH which is employed for simulating the macroscopic behavior at a coarse scale. Subsequently, Section 8.3 describes how the base simulation is refined in order to efficiently simulate and render fine-grained material. Finally, the effectiveness of the proposed pipeline is analyzed in Section 8.4.

8.2 Low-resolution Simulation

For computing the coarse simulation, we build on the SPH-based continuum method proposed in [AO11] which is described in Section 8.2.1 and Section 8.2.2. Section 8.2.3 discusses how the convergence of the algorithm can be improved by incorporating a more versatile treatment of the interface with solid boundaries. The description of the base simulation is completed by explaining how to simulate transitions from dry to wet material in Section 8.2.4.

8.2.1 Forces

Granular flow is governed by unilateral incompressibility which is described by two inequality constraints $\rho \leq \rho_0$ and $p \geq 0$. Accordingly, pressure values p are not negative and the material can not be compressed beyond ρ_0. These constraints can be directly plugged into any incompressible SPH solver by clamping negative pressures to zero. The resultant pressure field is then used to compute the pressure forces as

$$\mathbf{F}_i^p = -m_i \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij}. \tag{8.1}$$

For simulating granular material, [AO11] introduces a friction model which minimizes relative velocities measured by the strain rate $\dot{\epsilon}$. The strain rate is a 3×3 matrix which is computed as $\dot{\epsilon} = 0.5 \left(\nabla \mathbf{v} + \nabla \mathbf{v}^T \right)$. The gradient of the velocity $\mathbf{v}_i$ is computed as a sum of tenor products with

$$\nabla \mathbf{v}_i = \sum_j V_j \nabla W_{ij} \mathbf{v}_j^T. \tag{8.2}$$

Then a frictional stress tensor $\hat{\mathbf{s}}$ is computed that dissipates the strain rate. In order to simulate material with different angles of repose θ, the Drucker-Prager yield criterion

$$\sqrt{\sum \hat{s}_{ij}^2} \leq p\sqrt{2} \sin\theta \tag{8.3}$$

is employed. Thereby, friction is limited by pressure, i.e., in the absence of pressure, the frictional stress is zero. The frictional forces are computed with

$$\mathbf{F}_i^f = -m_i \sum_j m_j \left(\frac{\hat{\mathbf{s}}_i}{\rho_i^2} + \frac{\hat{\mathbf{s}}_j}{\rho_j^2} \right) \nabla W_{ij}. \tag{8.4}$$

8.2.2 Implementation

In order to compute (8.1) and (8.4), stress and pressure values have to be determined. For this, we employ the predictive-corrective method presented in [AO11], which is an adaption of the PCISPH method [SP09]. In this method, the constraint forces, i.e., pressure and friction forces, are iteratively computed in a Jacobi-like manner as described in the following.

Pressure

In each iteration l, pressure is related to the predicted density deviation at time $t + \Delta t$ using an equation of state (EOS) as

$$p_i^l(t) = \sum_l \gamma \Big(\max\left(0, \rho_i^l(t + \Delta t) - \rho_0\right) \Big), \tag{8.5}$$

where γ is a scaling factor which can be precomputed for a prototype particle with full neighborhood i as

$$\gamma = \frac{\rho_0^2}{2m_i^2\Delta t^2 \sum_j \nabla W_{ij}^T \nabla W_{ij}}. \tag{8.6}$$

$\rho_i^l(t+\Delta t)$ denotes the predicted density of particle i in iteration l which is computed as

$$\rho_i^l(t+\Delta t) = \sum_j m_j W\left(\mathbf{x}_i^l(t+\Delta t) - \mathbf{x}_j^l(t+\Delta t), h\right). \tag{8.7}$$

The predicted positions $\mathbf{x}_i^l(t+\Delta t)$ are computed with

$$\mathbf{x}_i^l(t+\Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t) + \Delta t^2 \frac{\mathbf{F}_i^{adv}(t) + \mathbf{F}_i^{p^{l-1}}(t) + \mathbf{F}_i^{f^{l-1}}(t)}{m_i}, \tag{8.8}$$

where $\mathbf{F}_i^{adv}(t)$ denotes non-constrained forces such as the gravity force. Please note that $\mathbf{F}_i^{p^{l-1}}(t)$ and $\mathbf{F}_i^{f^{l-1}}(t)$ are constraint forces which use implicit information, $\mathbf{F}_i^{adv}(t)$ uses exclusively information available at time t and does not change during the iterations. Employing the momentum preserving discretization (8.1), we can compute the pressure force in iteration l as

$$\mathbf{F}_i^{p^l}(t) = -m_i \sum_j m_j \left(\frac{p_i(t)^l}{\rho_i(t)^2} + \frac{p_j(t)^l}{\rho_j(t)^2}\right) \nabla W_{ij}(t). \tag{8.9}$$

Friction

Frictional forces constrain the strain rate to be below a user-defined threshold. The discretization (8.4) is a momentum preserving SPH approximation of $\nabla \cdot \hat{\mathbf{s}}_i$. Like for pressure, the unknown stress tensors are iteratively computed in a predictive-corrective manner. Thereby, stress is locally related to the predicted strain rate $\dot{\epsilon}_i^l(t+\Delta t)$ with

$$\mathbf{s}_i^l = \mathbf{s}_i^{l-1} + \mathbf{D}^{-1}\dot{\epsilon}_i^l(t+\Delta t), \tag{8.10}$$

where $\mathbf{D}$ denotes a diagonal matrix which can be precomputed for a prototype particle with full neighborhood

$$\mathbf{D} = \frac{\rho_0^3}{2m_i^2\Delta t} \sum_j \nabla W_{ij} \nabla W_{ij}^T. \tag{8.11}$$

In (8.10), the full stress tensor $\mathbf{s}$ is given. However, as in [NGL10, AO11], friction is expressed by the traceless deviatoric part $\hat{\mathbf{s}}$, only. Furthermore, in order to simulate different angles of repose, the yield criterion (8.3) has to be satisfied. Employing a piecewise approximation of (8.3) on each matrix component $s_i(ab)$ as in [NGL10], the frictional stress is computed as

$$\hat{\mathbf{s}}_i^l = \mathbf{s}_i^l - \frac{1}{3}\text{trace}\left(\mathbf{s}_i^l\right)\mathbf{I}_{3\times 3} \quad \text{with} \quad \left\|\hat{s}_i^l(ab)\right\| \leq p_i^l\sqrt{2/3}\sin\theta, \tag{8.12}$$

where $\mathbf{I}_{3\times 3}$ is the identity matrix. In each iteration, the predicted strain rate is computed as

$$\dot{\epsilon}_i^l(t+\Delta t) = 0.5\left(\nabla\mathbf{v}_i^l(t+\Delta t) + \left(\nabla\mathbf{v}_i^l(t+\Delta t)\right)^T\right), \tag{8.13}$$

where the predicted velocities in each iteration are evaluated as

$$\mathbf{v}_i^l(t+\Delta t) = \mathbf{v}_i(t) + \Delta t\frac{\mathbf{F}_i^{adv}(t) + \mathbf{F}_i^{p^{l-1}}(t) + \mathbf{F}_i^{f^{l-1}}(t)}{m_i}. \tag{8.14}$$

Stabilization

In this implementation, pressure and frictional constraint forces are computed simultaneously using a local approach. The iterative loop can be terminated when the maximum density $\max_i \rho_i(t+\Delta t)^l$ is below a threshold η and the maximum variation of the stress tensor is below the threshold ζ. However, the constraint forces are interdependent. This interdependency influences the convergence in low energy regions which prevents stable formations of sand piles, i.e., sliding does not stop even for large angles of repose. This effect is even more significant if a third constraint force is employed such as direct-forcing [BIT09] to handle interactions with boundaries. In order to simulate stable formations of sand piles, we suggest to apply the well-known concept of rigid-body sleeping [Sch02] to SPH particles. Thereby, we do not integrate SPH particles in time as long as their velocity is lower than a user-defined *sleep velocity*. Particles wake up automatically when their net-acceleration exceeds a critical value. We found that this is already sufficient to stop unwanted sliding at the free surface, but not at the contact region with solid boundaries. At interfaces with solid objects, additional considerations have to be employed. The next section addresses this issue.

8.2.3 Solid Body Interaction

The forces given in the previous section are internal forces, acting inside the material. At interfaces with solid objects, additional considerations have to be employed in order to guarantee non-penetration of objects and to simulate two-way coupled interactions with variable friction.

The direct forcing method [BIT09] used in [AO11] corrects predicted penetrations by constraining positions to the rigid surface. Different slip conditions are realized by directly manipulating particle velocities. This model does not conserve momentum and leads to significant oscillations in the pressure field as shown for fluids in [AIS+12]. Additionally, for the granular model described in Section 8.2.1, the noisy pressure field results in discontinuities of the frictional forces (8.4). In order to avoid perceivable artifacts, a small time step has to be employed and/or the number of iterations for correcting density errors has to be set high, i.e., larger than five.

Algorithm 6 Granular PCISPH solver. Coarse simulation model.

```
procedure UPDATE NEIGHBORHOOD
    for all particle i do
        find neighbors
procedure PREDICT ADVECTION
    for all particle i do
        apply gravity and material viscosity
        reset pressure and stress
procedure PRESSURE SOLVE
    k = 0
    while k < 3 do
        for all particle i do
            predict velocity and position
        for all particle i do
            predict density ρ*_i(t + Δt)
            if ρ*_i(t + Δt) > ρ_max then
                predict density error and strain rate
                increment stress and pressure
                apply yield condition
            else
                add discrete particle forces
        for all particle i do
            compute pressure force (8.1) and (8.16)
            compute friction force (8.4) and (8.20)
        k = k + 1
procedure INTEGRATION
    for all particle i do
        update velocity and position
```

We improve the robustness and the quality of the simulation by adapting the boundary handling method of [AIS+12] to interactions with granular material. Thereby, the surface of rigid objects is sampled with particles as in [BYM05]. Boundary particles contribute to the density of a granular particle. Accordingly, the density is computed as

$$\rho_i = \sum_j m_j W(\mathbf{x}_i - \mathbf{x}_j, h) + \sum_b \frac{\rho_0}{\delta_b} W(\mathbf{x}_i - \mathbf{x}_b, h), \tag{8.15}$$

where $\delta_b = \frac{1}{\sum_j W_{bj}}$ is the number density of a boundary particle b, summed up over all boundary particle neighbors j. $\Psi_b(\rho_0) \equiv \frac{\rho_0}{\delta_b}$ scales the contributions of a boundary particle according to the local boundary sampling and the reference density of the granular material. Thereby, (8.15) estimates the density correctly at boundaries for arbitrary samplings and material densities. Non-penetration is realized via pressure forces, acting from a boundary particle b on a granular particle i with

$$\mathbf{F}^p_{i \leftarrow b} = -m_i \Psi_b(\rho_0) \frac{p_i}{\rho_i^2} \nabla W_{ib}. \tag{8.16}$$

A detailed derivation can be found in [AIS+12].

We extend this model to interactions with granular materials by adding contributions of the

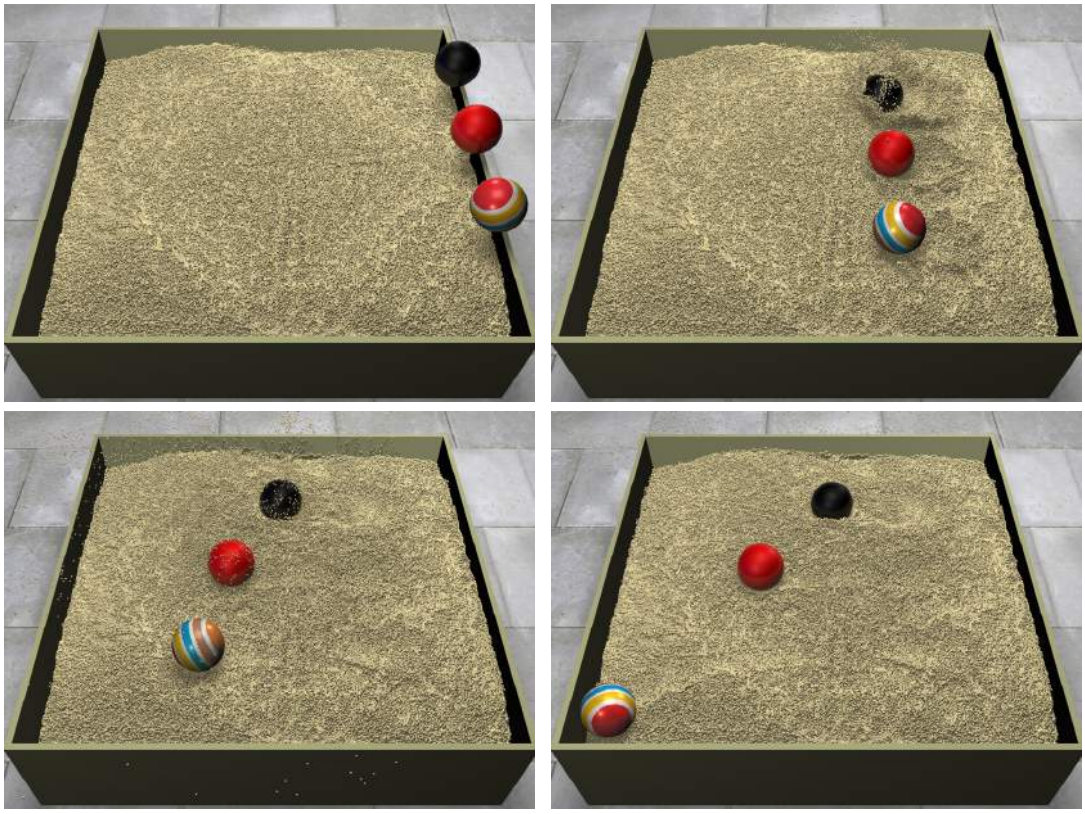

Figure 8.2: Demonstration of the proposed two-way coupling. Three spheres with different masses are tossed into a sand pool. Due to the friction-based coupling, the sand supports also the heaviest sphere (black) which is three times more dense than the sand.

boundary when computing frictional stresses. Therefore, we modify (8.2) to

$$\nabla \mathbf{v}_i = \sum_j V_j \nabla W_{ij} \mathbf{v}_j^T + \sum_b \frac{1}{\delta_b} \nabla W_{ib} \mathbf{v}_b^T. \tag{8.17}$$

The reformulation of (8.4) can be derived similar to (8.16) as

$$\mathbf{F}_{i \leftarrow b}^{f_s} = -m_i \Psi_b(\rho_0) \frac{\mathbf{s}_i}{\rho_i^2} \nabla W_{ib}. \tag{8.18}$$

However, (8.18) depends on the properties of the granular material, i.e., angle of repose and local pressure, but not on the material properties of the rigid object. In order to model external friction forces, we additionally employ the artificial viscosity model given in [AIS+12] which is written as

$$\mathbf{F}_{i \leftarrow b}^{v} = -m_i \Psi_b(\rho_0) \Pi_{ib} \nabla W_{ib}, \tag{8.19}$$

where $\Pi_{ib} = -\frac{\sigma_{ib} h c_s}{2\rho_i} \left(\frac{\min(\mathbf{v}_{ib} \cdot \mathbf{x}_{ib}, 0)}{|\mathbf{x}_{ib}|^2 + \epsilon h^2} \right)$. Here, c_s denotes the speed of numerical propagation and σ_{ib} controls the friction between the rigid object and the granular material. Applying the sum of (8.18) and (8.19) might overshoot the desired dissipative effect. Therefore,

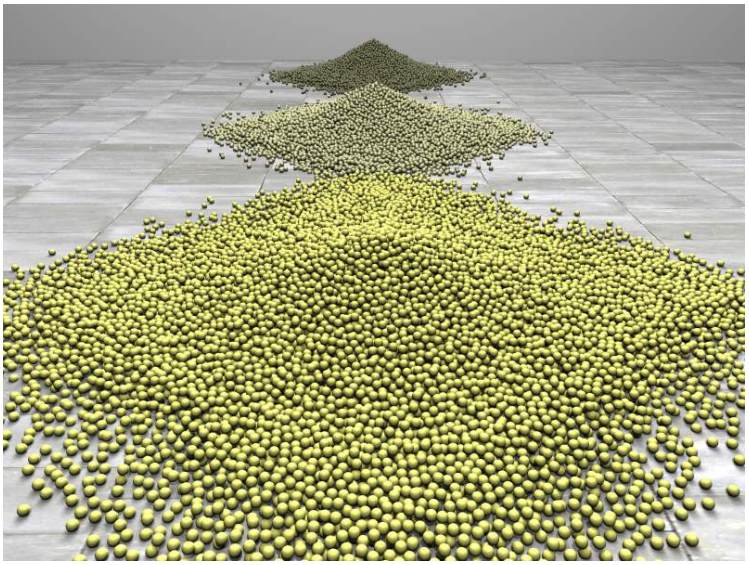

Figure 8.3: Sand piles with different angles of repose 30°, 45° and 60° (front to back).

we combine both forces by picking the one which maximizes dissipation. Accordingly, the dissipative contribution of a boundary particle b is computed as

$$\mathbf{F}^{f}_{i\leftarrow b} = \max(\left\|\mathbf{F}^{v}_{i\leftarrow b}\right\|, \left\|\mathbf{F}^{f_s}_{i\leftarrow b}\right\|). \tag{8.20}$$

Algorithm 6 outlines the steps performed in each simulation update of the base solver. Two-way coupling is easily realized by applying pressure (8.16) and friction (8.20) forces symmetrically to the boundary, see Figure 8.2.

8.2.4 Material Properties

The simulation model described so far is applicable to simulations of granular material with different angles of repose, see Figure 8.3. We further extend the granular simulation model to allow for simulations of dry and wet sand with controllable scaling. In dry sand volumes, air voids can form between grains, which reduces friction and allows the material to disperse freely. In contrast, moist sand volumes exhibit a higher friction in between grains which results in a sticky behavior. As a consequence, compared to dry material, a larger force needs to be exerted to make the wet material flow. This phenomenon can not be simulated by the frictional force (8.4) alone as this force acts only in tangential direction. In [AO11], the model of Gascon et al. [GZO10] is adapted in order to simulate cohesive effects as a function of the norm of the strain rate. However, this model imposes a severe restriction on the time step as mentioned in [AO11]. In contrast, we found that the surface tension model for SPH fluids presented in [BT07] is well suited to model cohesive effects for granular material without imposing restrictions on the time step. Thereby, surface tension is modeled as a sum of pairwise forces defined as

$$\mathbf{F}^{cohesion}_{i} = -\kappa \sum_{j} m_j W_{ij} \mathbf{x}_{ij}, \tag{8.21}$$

Figure 8.4: Simulation of dry sand (left) and wet sand (right). For both volumes the angle of repose is 50°. The materials differ in the cohesion intensity and the material viscosity. The wet volume has a cohesion intensity of 0.7 and the viscosity is set to 0.3. The cohesion intensity of the dry volume was set to zero and the viscosity was set to a minimal value of 0.01.

where κ controls the cohesion intensity. In order to model higher friction for wet volumes, we propose to increase the artificial viscosity constant of the granular material. We found that this gives better control and is numerically more robust than increasing the angle of repose.

The effectiveness of this model to simulate wet material is demonstrated on a simple scenario, where we simulated a dry and a wet volume, see Figure 8.4. The volumes differ only in the cohesion intensity and the viscosity constant. The dry material is simulated with no cohesion and an artificial viscosity constant of 0.01. For the wet volume, the cohesion intensity is set to 0.7 and the viscosity to 0.3. In order to simulate wetting processes when coupling the granular material with an SPH fluid, the transition from dry to wet sand can be realized by just increasing the cohesion intensity and the viscosity constant of the granular material.

8.2.5 Parameters

For the SPH interpolations, we use the cubic spline kernel [Mon05]. The support radius h is chosen as two times the average particle distance d_0 for the material at rest. Accordingly, we precompute the mass of each SPH particle with $m = V\rho_0$, where $V = \frac{1}{6}\pi d_0^3$ is the particle volume of a sand grain. The yield density ρ_0 is user given and can vary for different materials, e.g., for dry sand, we set ρ_0 to 1602 kg/m^3.

8.3 High-resolution Simulation

The simulation model described in the previous section employs a continuum approach which describes the granular flow at a macroscopic scale. Thereby, a simulation particle should be interpreted as a clump of matter and not as a single grain. Indeed, setting up the particle size to the real size of a sand grain, which is only a fraction of a millimeter, is prohibitive. This would not only explode the memory and computational costs, but also restrict the time step due to the CFL condition. Instead, we propose to simulate the material on a coarse scale and apply a secondary simulation with a set of highly resolved particles which can be directly used for rendering.

8.3.1 Sampling

Each time a low-resolution (LR) particle with radius r_{LR} is added, we sample its volume with HR particles. The initial sampling is crucial as it could easily introduce aliasing or distortions. Sampling the spherical volume of a particle leads to gaps while sampling the bounding box might cause staircase patterns. In order to avoid aliasing, we do not only generate HR samples inside the bounding box of the LR particle but also slightly outside, employing a distribution that prefers samples that are inside the LR volume.

Therefore, we divide the bounding box of each LR particle into seven support points, one at the particle center and one at each intersection point of the bounding box with the spherical particle volume. HR particles are randomly sampled around each support point in a cubical volume with length $2r_{LR}$. As the seven sample volumes overlap inside the LR volume, this strategy generates three times more HR samples inside the bounding box of the base particle than outside.

8.3.2 Advection

We derive the advection of HR particles from the following principles: HR particles should follow the mechanical flow that is given by the base simulation, but also should be allowed to disperse freely. Further, they should smoothly align with the surface of the base simulation without forming perceivable clumps. Finally, in order to guarantee efficient updates at large time steps, the advection method should not compute internal forces or perform collision tests between HR particles.

Alduan et al. [ATO09] set similar requirements to their model. They map the mechanical behavior by interpolating the velocity of LR particles to HR particles for advection. In order to avoid clumping, particles having one or no LR particle neighbors within the influence radius h_{HR} are only influenced by external forces. However, this simple distinction avoids clumping only if $h_{HR} \approx r_{LR}$. On the other hand, larger values of h_{HR} are required for smooth interpolations of the velocities. In all our experiments, we set $h_{HR} = 3r_{LR}$.

In contrast to [ATO09], we do not employ an explicit distinction of two cases, but propose a weighting that automatically and smoothly blends the contributions of the base simulation and external forces. Therefore, for each HR particle at position $\mathbf{x}_i$, we first compute distance-based weights w as

$$w(d_{ij}) = \max\left[0, \left(1 - \frac{d_{ij}^2}{h_{HR}^2}\right)^3\right], \tag{8.22}$$

where $d_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ is the distance between HR particles and LR simulation or boundary particles j in the support radius h_{HR}. (8.22) is a well-shaped kernel function which smoothly drops to zero. It is typically applied for reconstructing smooth surfaces of particle data [ZB05, AIAT12]. We employ it for computing the average velocity as

$$\mathbf{v}_i^*(t + \Delta t) = \frac{1}{\sum_j w(d_{ij})} \sum_j w(d_{ij}) \mathbf{v}_j. \tag{8.23}$$

As long as the number of LR samples is sufficient, (8.23) interpolates the velocity well. However, in sparsely sampled regions it results in visual clumping. In order to achieve smooth alignments of HR particles without perceivable clumps, we compute the velocities of HR particles as the sum of weighted external forces $\mathbf{F}^g$ and the interpolated velocity with

$$\mathbf{v}_i(t + \Delta t) = (1 - \alpha_i)\mathbf{v}_i^*(t + \Delta t) + \alpha_i \left(\mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}^g}{m} \right), \tag{8.24}$$

where α is non-zero in sparse regions only and increases with higher distances of $\mathbf{x}$ to the center of LR particles. It is defined as

$$\alpha_i = \begin{cases} 1 - \max_j w(d_{ij}) & \frac{\max_j w(d_{ij})}{\sum_j w(d_{ij})} \geq 0.6, \\ 1 - \max_j w(d_{ij}) & \max_j w(d_{ij}) \leq w(r_{LR}), \\ 0 & \text{otherwise.} \end{cases}$$

Here, the constant 0.6 is an empirically tested value which gives the best results when $h_{HR} = 3r_{LR}$. Finally, the position is integrated as

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t). \tag{8.25}$$

Accordingly, external forces are automatically applied in regions where clumping potentially occurs. Contributions of external forces are smoothly faded in and out, which on one hand allows HR particles to disperse freely and on the other hand results in smooth alignment with the materials surface, see Figure 8.5. Our model does faithfully upscale scenes with dynamic objects, as we take the positions and velocities of moving objects into account when interpolating the velocities.

8.4 Results

We present several scenarios which show different aspects of our approach including one-way and two-way solid body interactions. Comparison to previous work and performance evaluation is also provided. Timings are given for a 12-core 3.46 GHz Intel i7 with 24 GB of RAM. We make use of all threads by incorporating the parallel algorithms and cache-efficient data structures described in Chapter 4. Images were rendered with mental ray v3.9.4 [NVI11]. The video sequences are encoded with 50 frames per second. Thus, one frame corresponds to 0.02s in the following discussions.

8.4.1 Solid Body Interaction

We demonstrate the effectiveness of the proposed friction and pressure forces applied from and to the boundaries on a simple test scenario, see Figure 8.3. In this scene, we simulated

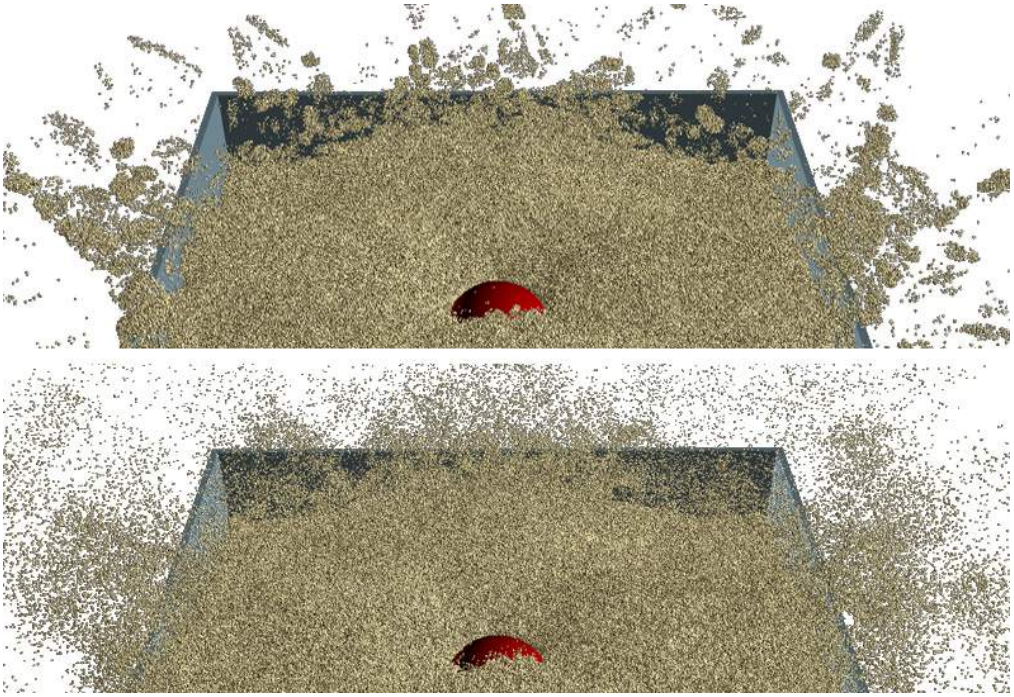

Figure 8.5: Upsampling comparison. In [ATO09], HR particles can not disperse freely if two or more base particles are in close proximity (top). Our method avoids clumping by weighting external and internal forces for all HR particles (bottom).

three stable piles with different angles of repose. We set the time step to 1 millisecond and used a fixed number of three iterations to compute pressure and stress values. For this setting, the proposed boundary handling enforces smooth pressure gradients at the boundary. In contrast, if we employ direct forcing as in [AO11], the density field oscillates which results in unnatural accelerations perceived as 'popping of particles' as is shown in the accompanying video.

Two-way coupling between rigid objects and granular material is demonstrated in Figure 8.2. Rigid objects interact differently with the granular material according to their density since the dissipation of energy for lighter objects is much less than for heavy objects. In contrast to fluids, granular material supports even objects with much higher material density than its own. This behavior is faithfully captured as the employed coupling is based on frictional stresses.

8.4.2 Upsampling

In order to show the benefit of the proposed refinement model, we compare it to [ATO09] on a scene where a heavy sphere hits the sand surface with high velocity, creating a splash. The base simulation sampled the material with 100K particles while the secondary simulation used 20.8M particles. Since in [ATO09], HR particles with more than one LR neighbor are not influenced by external forces, cluster of HR particles move uniformly in splash regions and at the sand surface, see Figure 8.5-top. In contrast, our method does not rely on complementary cases to compute the velocity, but weights external forces and base velocities using a well shaped kernel function. Accordingly, HR particles are allowed to disperse freely,

Figure 8.6: A bulldozer drives through a sand pile. The base simulation with 38K particles is shown on the left. The secondary simulation with 19.4 million HR particles on the right. The proposed model handles interactions with scripted and simulated rigid objects. By upsampling, visual detail is added which is not captured by the base simulation.

see Figure 8.5-bottom.

Opposed to previous work, we show that HR particles interact smoothly with moving objects, e.g., a bulldozer, see Figure 8.6. This is realized by taking positions and velocities of boundary particles into account when interpolating the velocity field. Furthermore, the proposed model can cope with very large upscaling factors. Figure 8.6 shows the refinement with an upscaling factor of 500. Please note that the largest value used in [ATO09] has been 38.

We allow particles to depart freely from the base simulation by computing the velocities of HR particles as the weighted sum of external forces and the interpolated velocity field. Thereby, HR particles uncover details that are not captured by the base simulation, see Figure 8.7.

8.4.3 Performance

The presented framework advances the efficiency compared to previous work in two ways. First, the employed boundary handling results in smooth pressure gradients which improves the robustness compared to [AO11]. Thus, larger time steps can be handled at a smaller

Figure 8.7: Sand is poured over a fixed rope-ladder. LR particles (top) do not slip through all gaps, in contrast to the finer-scaled HR particles (bottom). The secondary simulation captures a pattern at the ground which is not resolved by the base simulation (left).

number of iterations for computing pressure and frictional forces. In all presented scenarios, the primary simulation was performed with a time step of 1 ms and a fixed number of three iterations. Thereby, we measured a speed up of up to 6 compared to [AO11].

Second, we employ the refinement as a post-processing step at a different temporal resolution. In all presented scenarios, the time step for the secondary simulation was set to 10 ms. As no interactions between HR particles are computed, each particle can be updated independently which permits a straightforward parallelization. It should be noticed that our implementation took on average 11 seconds (1.4s LR + 9.6s HR) per frame for a scene with 19.4 million particles, see Table 8.1. In contrast, Alduan et al. [ATO09] reported an update rate of 5.5 minutes for 1.6 million HR particles and the same number of LR particles. Although this comparison does not take the respective hardware configurations into account, it indicates the efficiency of the proposed model.

8.4.4 Resolution Scaling

The effect of the frictional force employed in this framework is not invariant to the spatial and temporal discretization. This is demonstrated on a simple scenario where a dry material with an angle of repose of 55° is simulated at three different resolutions, see Figure 8.8. The coarse simulation with 7K particles and a radius of 0.05m ran at a time step of 1ms. The same time step could be set for the medium resolved simulation with 60K particles. The high-resolution simulation with 500K particles required a much smaller time step of 0.1ms.

	# particles		time / frame	
	LR	HR	LR	HR
Spheres (Figure 8.2)	96K	11.3M	3.1 s	5.8 s
Splash (Figure 8.5)	83K	20.8M	2.9 s	10.3 s
Sand-Piles (Figure 8.3)	45K	-	1.6 s	-
Bulldozer (Figure 8.6)	38K	19.4M	1.4 s	9.6 s
Rope-Ladder (Figure 8.7)	138K	17.3M	4.6 s	8.3 s

Table 8.1: Performance measurements for the given scenarios.

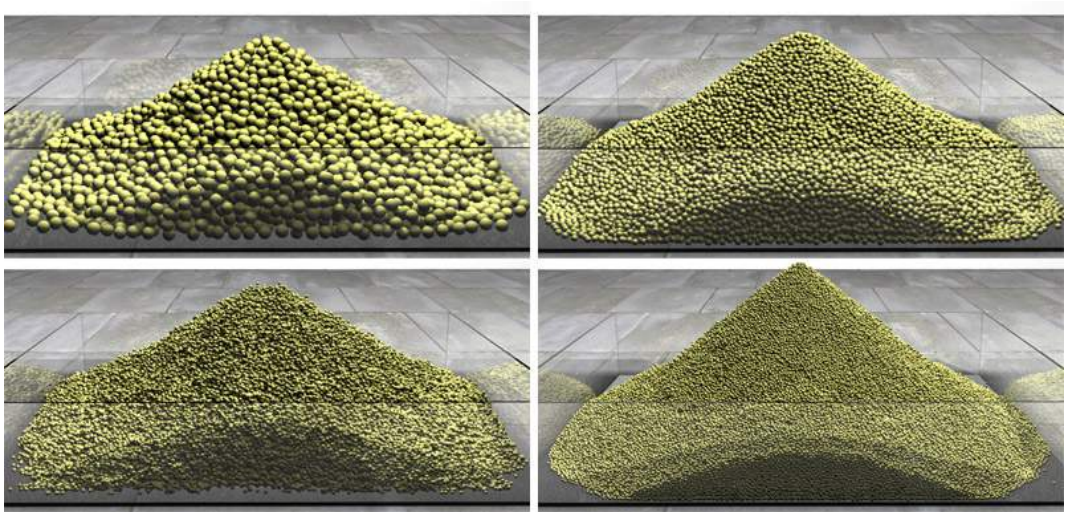

Figure 8.8: Base simulation with different resolutions. A coarse simulation of 7K particles with radius $r = 5cm$ (top-left), 60K particles with $r = 2.5cm$ (top-right) and a highly resolved simulation using 500K particles with $r = 1.25cm$ (bottom right). The bottom left image shows the refined coarse simulation with 500k secondary particles.

The rest angles of the sand piles obtained by the simulation were 31° (7K), 35° (60K) and 45° (500K). Thus, none of the simulations obtained the predefined angle of repose of 55°. Although this series indicates that the simulated angle of repose might converge to the desired value, this could not be verified as we could not find a practical time step for higher resolutions.

Due to the difference in the tolerated time step size, we measured a huge difference in the overall performance. One video frame for the different simulations could be computed in 0.3s (7K), 2.6s (60K) and 98s (500K). In order to evaluate the quality of the proposed refinement scheme, we also compare the simulation result of the highly resolved base simulation with the upsampled coarse simulation with 500K secondary particles, see Figure 8.8, bottom. The secondary simulation took only 0.8s per frame to refine the 7K simulation. Thus, in total it took 1.1 seconds (0.3s base simulation + 1.1s upsampling) to compute a frame with the proposed pipeline. This is almost 90 times faster than the time required for the highly resolved base simulation. The perceivable difference between the two simulations can be attributed to the differences in the base simulation (7K vs 500K) where we measured a deviation of 14° for the obtained angle of repose. On the other hand, this simple scenario is challenging for the secondary simulation as the movement of the granular material is slow

and nearly uniform. Furthermore, the base simulation is very coarsely sampled with 7K particles. For this configuration, clustering in the refined simulation is perceivable.

8.5 Discussion

In this chapter, an efficient framework for computing high-resolution simulations of granular material using a pure Lagrangian method has been proposed. Performance-critical forces are computed in a primary simulation on a relatively coarse scale. It has been shown how to realize stable and realistic interactions with rigid bodies by employing pressure and friction-based boundary forces. The stability of the base simulation can be significantly improved by incorporating the concept of rigid-body sleeping. This also stops unwanted sliding of sand piles. Furthermore, the simulation of dry and wet sand with controllable scaling has been realized.

Visual detail is added in a secondary simulation where high-resolution particles are coupled to the base flow. For advecting secondary particles, a smooth weighting of external forces with the velocity field of the base simulation has been proposed. This technique adds detail that is not captured otherwise while clumping of high-resolution particles is avoided, even for very large upscaling factors. Finally, implementation aspects and parameter setting have been discussed in detail.

The proposed coupling does not model static friction correctly which prohibits the animation of some interesting scenes, e.g., an accelerating dumper truck filled with sand. In such a scenario, the proposed model fails to keep the sand pile at rest when the truck starts moving.

Furthermore, in the refinement model, HR particles never interact with each other which makes it very efficient to compute on one hand. However, this might cause compression artifacts in sparsely sampled LR regions, e.g., at the edges of a sand pile, as HR particles are attracted to the ground by gravity. As a simple solution, interactions between HR particles could be computed in relevant regions employing either a discrete or a continuum model. Alternatively, this issue might be addressed by finding an adequate extension to the proposed interpolation heuristic.

Massive conglomerations of granular material, e.g., a beach, are more efficiently represented by heightfields [HBK09, KBKS09]. However, in these methods, the level of detail is limited as only two-dimensional information is mapped onto the three-dimensional space. Dispersion effects or the sliding of single grains can not be captured with such a representation.

In future work, the granular material could be coupled with an SPH fluid simulation in order to animate erosion effects and transitions from dry to moist sand and mud. In this context, it might be beneficial to investigate whether SPH fluid simulations can benefit from the proposed refinement model.

CHAPTER 9

Conclusion

This thesis has focused on performance aspects of incompressible fluid simulations with SPH. In this context, various methods have been presented which in sum allow to simulate large-scale simulations of water with bubbles, spray and foam. Therefore, a new pressure-projection solver has been proposed which outperforms previous SPH formulations such as PCISPH and ISPH by a factor of 6 and 7, respectively. In contrast to previous projection schemes, the proposed IISPH solver scales linearly with the size of the domain. The time step can be adaptively set with respect to the CFL condition, i.e., without rolling back the simulation as required in PCISPH. As the solver builds purely on SPH and does not compromise generality, it can be easily incorporated into multi-scale and multi-phase frameworks. In this context, the thesis discussed many important aspects for analyzing the performance in SPH. E.g., the performance of an SPH implementation is not just the relation of the number of particles to the computation time. The volume represented by each particle, the turbulence, the level of compression have to be also taken into account as they do influence the practical time step of any SPH solver due to the CFL condition.

Another performance-critical aspect in SPH is the querying and processing of interacting particle sets. In this thesis, two spatial acceleration structures have been proposed which are designed for many-core architectures. In order to yield a close to optimal scaling on parallel machines, a low memory traffic has to be enforced. This is achieved by mapping spatial locality onto the memory via a space-filling curve. In the context of SPH, the performance of the presented data structures has been thoroughly analyzed and compared with three existing variants of uniform grids, i.e., basic uniform grid, spatial hashing and index sort. The proposed structures can be considered state-of-the art for proximity queries in SPH and have been incorporated into GPU-based SPH simulations, e.g., [OK12, MM13]. Furthermore, they have been successfully employed to speed up the surface reconstruction of SPH simulations in [AIAT12].

The performance and robustness of the simulation is also influenced by the handling of interfaces with solids. Special considerations have to be taken into account at the interface with solid objects in order to prevent spatial and temporal discontinuities of physical properties which could lead to perceivable simulation artifacts and numerical instabilities. This thesis has presented a pressure-based boundary scheme which enforces smooth quantity fields at the fluid-solid interface. Non-penetration of arbitrary shaped boundaries is enforced even for comparatively large time steps. As the proposed boundary-handling scheme couples the fluid and the solid via the pressure field, the incorporation into fluid solvers like the proposed IISPH method is straightforward.

Many prominent effects in a turbulent flow arise from the interplay of air and water. However, modeling these effects is numerically challenging due to the high density ratio of air

to water and is often neglected in computer graphics. This thesis has presented an efficient model to capture phenomena like air bubbles, foam and spray. For small-scale simulations such as pouring water into a glass, a two-way coupled air-water model is proposed. High density ratios can be robustly simulated by coupling the two phases separately via the velocity field. This is sufficient to capture realistic drag effects. For large-scale simulations, the influence of air onto water might be neglected and the model could be applied in a post-process. Air entrapment and the formation of air-water mixtures is simulated without explicitly modeling the air surrounding the fluid by generating air bubbles, spray and foam on the fly. The model overcomes stability issues, can handle large time steps and is efficient to compute. Simulating the formation and transport of air-water mixtures adds prominent visual detail to small and large-scale fluid simulations. The proposed model is fully parallelized and adds only little computational overhead. Accordingly, it can be well employed for interactive applications like games as has been demonstrated by Macklin and Mueller in [MM13].

Finally, this thesis has presented a multi-resolution framework, where numerically critical forces are computed on a coarse scale while visual detail is added in a secondary simulation. Thereby, comparatively large time steps could be used. Using the example of sand, it has been shown that the proposed pipeline is effective in generating compelling animations of sand including the two-way interaction with rigid objects.

9.1 Future Work

Based on the presented contributions, there are plenty of possible directions for future work to further improve the performance and quality of SPH simulations.

For example, the particle deficiency at the free surface still poses some challenges to the SPH concept. As the interpolation domain is not sufficiently sampled, the density summation approach leads to significant approximation errors. This in turn might affect the pressure and force field. Common strategies are to either correct density values that are lower than the rest density using a Shepard filter or to simply assume underestimated densities to be equal to the rest density of the fluid. A promising strategy is to sample the free surface with ghost particles which contribute to the density computation [SB12]. However, it remains not clear if the proposed re-sampling of ghost particles might affect the stability of incompressible SPH implementations as the ghost-particle method has not yet been incorporated into an incompressible SPH solver. Solving the particle deficiency problem at the free surface should eliminate the exaggerated cohesion effects of the conjugate gradient solver for IISPH which has been presented in Chapter 3.5.2. This could improve the efficiency of IISPH as the convergence rate of conjugate gradient is superior to the employed relaxed Jacobi method.

Furthermore, it should be investigated if the IISPH formulation allows for an alternative multi-scale approach. In current multi-resolution methods for SPH, e.g. [APKG07, SG11, OK12], the transition between the resolutions requires sophisticated methods in order to keep the simulation stable. There is good reason to believe that the velocity projection employed in IISPH could ease the handling of and transitions into interfaces of varying resolutions. In such a framework, the resolution might be automatically varied either according to view dependent criteria or with respect to the dynamics of the flow. This should allow for practical simulations of very large domains at high resolution.

Finally, a comparison of IISPH with grid-based solvers could be beneficial. Please note that the techniques presented in this thesis allow for simulations of large bodies of water, although pure particle-based methods have been considered not to be practical for such simulations.

The reason is that solving the equations of motions on an unstructured and dynamic grid has been considered to be computationally too expensive in comparison to solving the equations on a static Cartesian grid. For grids, spatial derivatives can be approximated via finite differences using a small set of sample points. However, as advecting the quantities on a grid poses some problems, methods which advect the quantities with particles are preferred. Among these methods, FLIP is currently considered the state-of-the art for large-scale simulations. In FLIP, the grid can be interpreted as an auxiliary structure which simplifies the pressure computation. Typically, the grid resolution is chosen as half the particle resolution. Thereby, in three dimensions, the number of pressure samples is reduced by factor eight. In contrast, in SPH, the pressure values are computed at full resolution. Comparing the performance of SPH and FLIP is therefore not straightforward and beyond the scope of this thesis. In future work, it has to be identified how the two methods relate to each other, how their performance can be compared, and how the incorporated approximations affect the simulation result.

Curriculum Vitae

Personal data

Name	Markus Ihmsen
Date of birth	Feb 06, 1980
Place of birth	Freiburg, Germany

Education

2008	Ph.D. candidate and research assistant, Albert-Ludwigs-University, Freiburg
2007	Diploma in Computer Science (Dipl.-Inf.)
2001-2007	Studies in Computer Science, Albert-Ludwigs-University, Freiburg
2000	German Abitur, Theodor-Heuss Gymnasium, Freiburg

Publications

[ICS+13] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, M. Teschner, **"Implicit Incompressible SPH"**. IEEE Transactions on Visualization and Computer Graphics. 2013, doi:10.1109/TVCG.2013.105.

[IWT13] M. Ihmsen, A. Wahl, M. Teschner, **"A Lagrangian Framework for Simulating Granular Material with High Detail"**. *Computers & Graphics*, vol. 38, no. 5, pp.1-10, 2013.

[IAAT12] M. Ihmsen, N. Akinci, G. Akinci, M. Teschner, **"Unified Spray, Foam and Bubbles for Particle-based Fluids"**. *The Visual Computer (Proc. CGI 2012)*, vol.28, no.6-8, pp. 669-677, 2012.

[IWT12] M. Ihmsen, A. Wahl, M. Teschner, **"High-Resolution Simulation of Granular Material with SPH"**. *Proc. VRIPHYS*, pp. 53-60, 2012. Best Paper Award.

[AIAT12] G. Akinci, M. Ihmsen, N. Akinci, M. Teschner, **"Parallel Surface Reconstruction for Particle-based Fluids"**. *Computer Graphics Forum*, vol. 31, no.6, pp. 1797-1809, 2012.

[AAIT12] G. Akinci, N. Akinci, M. Ihmsen, M. Teschner, **"An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids"**. *Proc. VRIPHYS*, pp. 61-68, 2012.

[AIS+12] N. Akinci, M. Ihmsen, B. Solenthaler, G. Akinci, M. Teschner, **"Versatile Rigid-Fluid Coupling for Incompressible SPH"**. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 62:1-62:8, 2012.

[IABT11] M. Ihmsen, N. Akinci, M. Becker, M. Teschner, **"A Parallel SPH Implementation on Multi-core CPUs"**. *Computer Graphics Forum*, vol. 30, no. 1, pp. 99-112, 2011.

[IBAT11] M. Ihmsen, J. Bader, G. Akinci, M. Teschner, **"Animation of Air Bubbles with SPH"**. *Proc. International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 225–234, 2011.

[GIT11] M. Gissler, M. Ihmsen, M. Teschner, **"Efficient Uniform Grids for Collision Handling in Medical Simulators"**. *Proc. International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 79–84, 2011.

[IAGT10] M. Ihmsen, N. Akinci, M. Gissler, M. Teschner, **"Boundary Handling and Adaptive Time-stepping for PCISPH"**. *Proc. VRIPHYS*, pages 79–88, 2010.

[BIT09] M. Becker, M. Ihmsen, M. Teschner, **"Corotated SPH for deformable solids"**. *Proc. Eurographics Workshop on Natural Phenomena*, pages 27–34, 2009.

APPENDIX A

SPH - Kernel

Generally, the kernel functions can be written as $W_h(q(\mathbf{x}))$ with $q(\mathbf{x}) = \frac{\|\mathbf{x}\|}{h}$. The bicubic spline kernel function is written as

$$W(\mathbf{x}, h) = W_h(q(\mathbf{x})) = \frac{\sigma}{h^d} \begin{cases} 6q^3 - 6q^2 + 1 & 0 \leq q \leq \frac{1}{2} \\ 2(1-q)^3 & \frac{1}{2} \leq q \leq 1 \\ 0 & q > 1 \end{cases}, \tag{A.1}$$

where d denotes the dimension of the simulation, $q = q(\mathbf{x})$ and

$$\sigma = \begin{cases} \frac{4}{3} & d = 1 \\ \frac{40}{7\pi} & d = 2 \\ \frac{8}{\pi} & d = 3. \end{cases}$$

Please note that the presented version is normalized with respect to the support radius h. In contrast, in [Mon92], the spline kernel is given for a support radius of $2h$ since in [Mon92], h denotes the rest distance of the particles and not the support radius.

A.1 Gradient - First Derivative

For computing the gradient of the kernel function $W_h(q(\mathbf{x}))$, we have to compute the partial derivatives of W with respect to x_1, x_2, x_3 where $\mathbf{x} = (x_1, x_2, x_3)^T$. Since h is not a variable, but a constant, W is not derived for h. Accordingly, the gradient is computed as

$$\nabla W = \frac{\partial W}{\partial x_1}\mathbf{e}_1 + \frac{\partial W}{\partial x_2}\mathbf{e}_2 + \frac{\partial W}{\partial x_3}\mathbf{e}_3. \tag{A.2}$$

In order to compute the partial derivatives, the chain rule has to be applied as

$$\frac{\partial W}{\partial x_i} = \frac{\partial W}{\partial q} \cdot \frac{\partial q}{\partial x_i}. \tag{A.3}$$

Note that $\frac{\partial W}{\partial q}$ is just the derivative of W with respect to q and $\frac{\partial q}{\partial x_i}$ is computed as

$$\begin{aligned}\frac{\partial q}{\partial x_i} &= \frac{\partial \sqrt{x_1^2 + x_2^2 + x_3^2} \cdot h^{-1}}{\partial x_i} \\ &= \frac{1}{2} \frac{2x_i}{\sqrt{x_1^2 + x_2^2 + x_3^2}} \frac{1}{h} \\ &= \frac{x_i}{\|\mathbf{x}\| h}.\end{aligned} \tag{A.4}$$

Thus, with (A.2) the gradient of q can be computed with

$$\nabla q = \frac{\mathbf{x}}{\|\mathbf{x}\| h}. \tag{A.5}$$

Putting it all together, ∇W computes to

$$\nabla W = \frac{\partial W}{\partial q} \nabla q. \tag{A.6}$$

.

Consequently, the gradient of (A.1) is defined as

$$\begin{aligned}\nabla W_h(q(\mathbf{x})) &= \frac{\partial W_h}{\partial q} \cdot \nabla q, \text{ with} \\ \nabla q &= \frac{\mathbf{x}}{\|\mathbf{x}\| h}, \text{ and} \\ \frac{\partial W_h}{\partial q} &= \frac{6\sigma}{h^d} \begin{cases} 3q^2 - 2q & 0 \leq q \leq \frac{1}{2} \\ -(1-q)^2 & \frac{1}{2} \leq q \leq 1 \\ 0 & q > 1. \end{cases}\end{aligned}$$

A.2 Laplacian - Second Derivative

In order to compute the Laplacian of W we have to derive the gradient. This results in a scalar value. We write the Laplacian as

$$\Delta W = \nabla^2 W = \nabla \left(\frac{\partial W}{\partial q} \nabla q \right), \tag{A.7}$$

where the partial derivative x_i is obtained as

$$\begin{aligned}\frac{\partial}{\partial x_i} \left(\frac{\partial W}{\partial q} \nabla q \right) &= \left(\frac{\partial}{\partial x_i} \frac{\partial W}{\partial q} \right) \cdot \nabla q + \frac{\partial W}{\partial q} \cdot \left(\frac{\partial}{\partial x_i} \nabla q \right) \\ &= \frac{\partial^2 W}{\partial q^2} \cdot \frac{x_i}{\|\mathbf{x}\| h} \cdot \nabla q + \frac{\partial W}{\partial q} \cdot \frac{\partial}{\partial x_i} \nabla q\end{aligned} \tag{A.8}$$

by first applying the product rule and then the chain rule.

Let us have a closer look at how the partial derivatives of ∇q are computed. For this we need to apply the quotient rule

$$\left(\frac{g}{f} \right)' = \frac{g' \cdot f - g \cdot f'}{f^2}. \tag{A.9}$$

Taking $g = \mathbf{x}$ and $f = \|\mathbf{x}\| h$, we get

$$\begin{aligned} \frac{\partial}{x_i} \nabla q &= \frac{\mathbf{e}_i \cdot \|\mathbf{x}\| h - \mathbf{x} \cdot \frac{x_i}{\|x\|} h}{(\|\mathbf{x}\| \cdot h)^2} \cdot \mathbf{e}_i \\ &= \frac{\|\mathbf{x}\| - \frac{x_i^2}{\|\mathbf{x}\|}}{\|\mathbf{x}\|^2 h} \\ &= \frac{\sum_j x_j^2 - x_i^2}{\|\mathbf{x}\|^3 h}. \end{aligned} \tag{A.10}$$

Accordingly, if we add all partial derivatives above, $\nabla^2 q$ simplifies to

$$\nabla^2 q = \frac{2 \|\mathbf{x}\|^2}{\|\mathbf{x}\|^3 h} = \frac{2}{\|\mathbf{x}\| h}. \tag{A.11}$$

Now, we can plug $\nabla^2 q$ into (A.8) and compute $\nabla^2 W$ as

$$\begin{aligned} \nabla^2 W &= \frac{\partial^2 W}{\partial q^2} \cdot \frac{\mathbf{x}}{\|\mathbf{x}\| h} \cdot \nabla q + \frac{\partial W}{\partial q} \cdot \nabla^2 q \\ &= \frac{\partial^2 W}{\partial q^2} \cdot \frac{\mathbf{x}}{\|\mathbf{x}\| h} \cdot \frac{\mathbf{x}}{\|\mathbf{x}\| h} + \frac{\partial W}{\partial q} \cdot \frac{2}{\|\mathbf{x}\| h} \\ &= \frac{\partial^2 W}{\partial q^2} \cdot \frac{1}{h^2} + \frac{\partial W}{\partial q} \cdot \frac{2}{\|\mathbf{x}\| h} \end{aligned} \tag{A.12}$$

Finally, the Laplacian of (A.1) computes to

$$\nabla^2 W_h(q(\mathbf{x})) = \frac{\partial^2 W_h}{\partial q^2} \cdot \frac{1}{h^2} + \frac{\partial W_h}{\partial q} \cdot \frac{2}{\|\mathbf{x}\| h} \text{ , where}$$

$$\frac{\partial^2 W_h}{\partial q^2} = \frac{6\sigma}{h^d} \begin{cases} 6q - 2 & 0 \leq q \leq \frac{1}{2} \\ 2(1-q) & \frac{1}{2} \leq q \leq 1 \\ 0 & q > 1. \end{cases}$$

Bibliography

[AAIT12] G. Akinci, N. Akinci, M. Ihmsen, and M. Teschner. An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids. In *Proceedings VRIPHYS*, pages 53–60, 2012.

[ABC+07] C. Ammann, D. Bloom, J. M. Cohen, J. Courte, L. Flores, S. Hasegawa, N. Kalaitzidis, T. Tornberg, L. Treweek, B. Winter, and C. Yang. The birth of sandman. In *ACM SIGGRAPH 2007 sketches*, 2007.

[AIAT12] G. Akinci, M. Ihmsen, N. Akinci, and M. Teschner. Parallel Surface Reconstruction for Particle-Based Fluids. *Computer Graphics Forum*, 32(1):99–112, 2012.

[AIS+12] N. Akinci, M. Ihmsen, B. Solenthaler, G. Akinci, and M. Teschner. Versatile Rigid-Fluid Coupling for Incompressible SPH. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 30(4):72:1–72:8, 2012.

[AIY+04] T. Amada, M. Imura, Y. Yasumoro, Y. Manabe, and K. Chihara. Particle-based fluid simulation on GPU. In *ACM Workshop on General-Purpose Computing on Graphics Processors*, pages 1–6, 2004.

[All12] A. M. Allah. *An Improved Incompressible Smoothed Particle Hydrodynamics to Simulate Fluid-Soil-Structure Interactions*. PhD thesis, Graduate School of Engineering Kyushu University, 2012.

[Amd67] G. Amdahl. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[AO11] I. Alduán and M. A. Otaduy. SPH granular flow with friction and cohesion. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 25–32, 2011.

[APKG07] B. Adams, M. Pauly, R. Keiser, and L. Guibas. Adaptively sampled particle fluids. In *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, volume 26, pages 48:1–48:7, 2007.

[ATO09] I. Alduán, A. Tena, and M. A. Otaduy. Simulation of High-Resolution Granular Media. In *Proceedings of Congreso Español de Informática Gráfica*, pages 1–8, 2009.

[Bic04] N. Bicanić. Discrete element methods. *Encyclopedia of Computational Mechanics*, 1, 2004.

[BIT09] M. Becker, M. Ihmsen, and M. Teschner. Corotated SPH for deformable solids. *Eurographics Workshop on Natural Phenomena*, pages 27–34, 2009.

[BLS12] K. Bodin, C. Lacoursière, and M. Servin. Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):516–526, 2012.

[BR86] J. Brackbill and H. Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2):314–343, 1986.

[Bri08] R. Bridson. *Fluid Simulation for Computer Graphics*. A K Peters / CRC Press, 2008.

[BSK+07] R. Bredow, D. Schaub, D. Kramer, M. Hausman, D. Dimian, and R. S. Duguid. Surf's up: the making of an animated documentary. In *SIGGRAPH 2007 courses*, pages 1–123, 2007.

[BSW10] F. Bagar, D. Scherzer, and M. Wimmer. A Layered Particle-Based Fluid Model for Real-Time Rendering of Water. *Computer Graphics Forum (Proceedings EGSR 2010)*, 29(4):1383–1389, 2010.

[BT07] M. Becker and M. Teschner. Weakly compressible SPH for free surface flows. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 209–217, 2007.

[BTT09] M. Becker, H. Tessendorf, and M. Teschner. Direct forcing for lagrangian rigid-fluid coupling. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):493–503, 2009.

[BYM05] N. Bell, Y. Yu, and P. J. Mucha. Particle-based simulation of granular materials. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer Animation*, pages 77–86, 2005.

[CL95] J. Chen and N. Lobo. Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations. *Graphical Models and Image Processing*, 57(2):107–116, 1995.

[CM10] N. Chentanez and M. Müller. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 197–206, 2010.

[CM11] N. Chentanez and M. Müller. Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 30:82:1–82:10, 2011.

[CNL+08] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y. kuang Chen, A. Baransi, and P. Dubey. Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture. In *34th Intel Conference on Very Large Data Bases*, pages 1313–1324, 2008.

[CPPK07] P. Cleary, S. Pyo, M. Prakash, and B. Koo. Bubbling and frothing liquids. *ACM Transaction on Graphics (Proceedings SIGGRAPH)*, 26(3):97, 2007.

[CR99] S. Cummins and M. Rudman. An SPH Projection Method. *Journal of Computational Physics*, 152(2):584–607, 1999.

[DC96] M. Desbrun and M.-P. Cani. Smoothed Particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76. Springer-Verlag, 1996.

[DK01] R. Dalrymple and O. Knio. SPH modeling of water waves. In *Proceedings Coastal Dynamics*, pages 779–787, 2001.

[ESE07] M. Ellero, M. Serrano, and P. Español. Incompressible smoothed particle hydrodynamics. *Journal of Computational Physics*, 226(1):1731–1752, 2007.

[FAMO99] R. P. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152(2):457–492, 1999.

[FAZ+00] S. Fangmeier, J. Anderson, H. Zargarpour, D. Smythe, and T. Alexander. Industrial Light + Magic : The Making of 'The Perfect Storm'. SIGGRAPH Panel, 2000.

[FE08] F. Fleissner and P. Eberhard. Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. *International Journal for Numerical Methods in Engineering*, 74:531–553, 2008.

[Fed02] R. Fedkiw. Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method. *Journal of Computational Physics*, 175(1):200–224, 2002.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings SIGGRAPH 2001*, pages 23–30, 2001.

[FG07] S. Falappi and M. Gallati. SPH Simulation of water waves generated by granular landslides. In *Proceedings of 32nd Congress of IAHR (International Association of Hydraulic Engineering & Research*, 2007.

[FM96] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[FR86] A. Fournier and W. T. Reeves. A simple model of ocean waves. In *Proceedings SIGGRAPH*, pages 75–84, 1986.

[FSJ01] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In *Proceedings SIGGRAPH*, pages 15–22, 2001.

[GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 22(3):871–878, 2003.

[GH04] S. T. Greenwood and D. H. House. Better with bubbles: enhancing the visual realism of simulated fluid. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 287–296, 2004.

[GIT11] M. Gissler, M. Ihmsen, and M. Teschner. Efficient Uniform Grids for Collision Handling in Medical Simulators. In *Proceedings of International Conference on Computer Graphics Theory and Applications GRAPP*, pages 79–84, 2011.

[GLR+06] W. Geiger, M. Leo, N. Rasmussen, F. Losasso, and R. Fedkiw. So real it'll make you wet. In *SIGGRAPH 2006 sketches*, 2006.

[GM77] R. Gingold and J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–398, 1977.

[Gre08] S. Green. Particle-based Fluid Simulation. "http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_ParticleFluids.pdf", 2008.

[GSLF05] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 24:973–981, 2005.

[GSSP10] P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola. Interactive SPH simulation and rendering on the GPU. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–64, 2010.

[GZO10] J. Gascón, J. S. Zurdo, and M. A. Otaduy. Constraint-Based Simulation of Adhesive Contact. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 39–44, 2010.

[HA06] X. Hu and N. Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *Journal of Computational Physics*, 213(2):844–861, 2006.

[HA07] X. Hu and N. Adams. An incompressible multi-phase SPH method. *Journal of Computational Physics*, 227(1):264–278, 2007.

[HBK09] D. Holz, T. Beer, and T. Kuhlen. Soil Deformation Models for Real-Time Simulation: A Hybrid Approach. In *Proceedings VRIPHYS*, pages 21–30, 2009.

[Hie07] S. Hieber. *Particle-Methods for Flow-Structure Interactions*. PhD thesis, Swiss Federal Institute of Technology, 2007.

[HK03] J.-M. Hong and C.-H. Kim. Animation of Bubbles in Liquid. *Computer Graphics Forum*, 22:253–262, 2003.

[HKK07a] T. Harada, S. Koshizuka, and Y. Kawaguchi. Sliced data structure for particle-based simulations on gpus. In *Proceedings of the Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 55–62, 2007.

[HKK07b] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed Particle Hydrodynamics on GPUs. In *Proceedings of Computer Graphics International*, pages 63–70, 2007.

[HLWW12] X. He, N. Liu, H. Wang, and G. Wang. Local Poisson SPH for viscous incompressible fluids. *Computer Graphics Forum*, 31:1948–1958, 2012.

[HLYK08] J.-M. Hong, H.-Y. Lee, J.-C. Yoon, and C.-H. Kim. Bubbles alive. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 27:48:1–48:4, 2008.

[HTK+04] B. Heidelberger, M. Teschner, R. Keiser, M. Mueller, and M. Gross. Consistent Penetration Depth Estimation for Deformable Collision Response. In *Proceedings Vision, Modeling, Visualization*, pages 339–346, 2004.

[HW65] F. Harlow and J. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface. *The Physics of Fluids*, 8:2182–2189, 1965.

[IAAT12] M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner. Unified Spray, Foam and Bubbles for Particle-based Fluids. *The Visual Computer*, 30(1):99–112, 2012.

[IABT11] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner. A Parallel SPH Implementation on Multi-core CPUs. *Computer Graphics Forum*, 30(1):99–112, 2011.

[IAGT10] M. Ihmsen, N. Akinci, M. Gissler, and M. Teschner. Boundary Handling and Adaptive Time-stepping for PCISPH. In *Proceedings VRIPHYS*, pages 79–88, 2010.

[IBAT11] M. Ihmsen, J. Bader, G. Akinci, and M. Teschner. Animation of Air Bubbles with SPH. In *Computer Graphics Theory and Applications GRAPP*, pages 225–234, 2011.

[ICS+13] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*, 2013. doi:10.1109/TVCG.2013.105.

[IGLF06] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transaction on Graphics (Proceedings SIGGRAPH)*, 25:805–811, 2006.

[IWT12] M. Ihmsen, A. Wahl, and M. Teschner. High-Resolution Simulation of Granular Material with SPH. In *Proceedings VRIPHYS*, pages 53–60. Best Paper Award, 2012.

[IWT13] M. Ihmsen, A. Wahl, and M. Teschner. A lagrangian framework for simulating granular material with high detail. *Computers & Graphics*, 38:1–10, 2013.

[JLL04] C. Josserand, P.-Y. Lagrée, and D. Lhuillier. Stationary shear flows of dense granular materials: a tentative continuum modelling. *The European Physical Journal E: Soft Matter and Biological Physics*, 14(6):127–135, 2004.

[KAD+06] R. Keiser, B. Adams, P. Dutré, L. Guibas, and M. Pauly. Multiresolution Particle-Based Fluids. Technical report, ETH Zurich, 2006.

[KAG+05] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutré, and M. Gross. A Unified Lagrangian Approach to Solid-Fluid Animation. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 125–134, 2005.

[KBKS09] P. Kristof, B. Benevs, J. Krivanek, and O. Stava. Hydraulic Erosion Using Smoothed Particle Hydrodynamics. *Computer Graphics Forum (Proceedings of Eurographics)*, 28(2), 2009.

[KCC+06] J. Kim, D. Cha, B. Chang, B. Koo, and I. Ihm. Practical Animation of Turbulent Splashing Water. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 335–344, 2006.

[KFCO06] B. Klingner, B. Feldman, N. Chentanez, and J. O'Brien. Fluid animation with dynamic meshes. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 25:820–825, 2006.

[KGS09] A. Khayyer, H. Gotoh, and S. Shao. Enhanced predictions of wave impact pressure by improved incompressible SPH methods. *Applied Ocean Research*, 31(2):111 – 131, 2009.

[KM90] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings SIGGRAPH*, pages 49–57, 1990.

[KS09] J. Kalojanov and P. Slusallek. A parallel algorithm for construction of uniform grids. In *Proceedings of the ACM conference on High Performance Graphics*, pages 23–28, 2009.

[KSK10] D. Kim, O.-Y. Song, and H.-S. Ko. A practical simulation of dispersed bubble flow. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 29:70:1–70:5, 2010.

[KVG02] H. Kück, C. Vogelgsang, and G. Greiner. Simulation and rendering of liquid foams. In *Proceedings of Graphics Interface*, pages 81–88, 2002.

[KW06] P. Kipfer and R. Westermann. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface*, pages 41–48, 2006.

[LAD08] T. Lenaerts, B. Adams, and P. Dutré. Porous flow in particle-based fluid simulations. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 27(3):49:1–49:8, 2008.

[LD08] A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)*, 27(4):1235–1244, 2008.

[LD09] T. Lenaerts and P. Dutre. Mixing fluids and granular materials. *Computer Graphics Forum*, 28(2):213–218, 2009.

[LGF04] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 23:457–462, 2004.

[LHM95] A. Luciani, A. Habibi, and E. Manzotti. A multi scale physical model of granular materials. In *Proceedings Graphics Interface*, pages 136–146, 1995.

[LKO05] J. Liu, S. Koshizuka, and Y. Oka. A hybrid particle-mesh method for viscous, incompressible, multiphase flows. *Journal of Computational Physics*, 202(1):65–93, 2005.

[LM93] X. Li and J. M. Moshell. Modeling soil: Realtime dynamic models for soil slippage and manipulation. In *Proceedings SIGGRAPH*, pages 361–368, 1993.

[LTKF08] F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw. Two-Way Coupled SPH and Particle Level Set Fluid Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):797–804, 2008.

[Luc77] L. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[MFZ97] J. Morris, P. Fox, and Y. Zhu. Modeling Low Reynolds Number Incompressible Flows using SPH. *Journal of Computational Physics*, 136(1):214–226, 1997.

[MK09] J. Monaghan and J. Kajtar. SPH particle boundary forces for arbitrary boundaries. *Computer Physics Communications*, 180(10):1811–1820, 2009.

[MM97] J. Morris and J. Monaghan. A Switch to Reduce SPH Viscosity. *Journal of Computational Physics*, 136(1):41–50, 1997.

[MM13] M. Macklin and M. Mueller. Position Based Fluids. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 32:1–5, 2013.

[MMS07] V. Mihalef, D. Metaxas, and M. Sussman. Textured Liquids based on the Marker Level Set. *Computer Graphics Forum*, 26:457–466, 2007.

[MMS09] V. Mihalef, D. N. Metaxas, and M. Sussman. Simulation of two-phase flow with sub-scale droplet and bubble effects. *Computer Graphics Forum*, 28(2):229–238, 2009.

[Mon92] J. Monaghan. Smoothed particle hydrodynamics. *Ann. Rev. Astron. Astrophys.*, 30:543–574, 1992.

[Mon94] J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110(2):399–406, 1994.

[Mon05] J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 2005.

[MSKG05] M. Müller, B. Solenthaler, R. Keiser, and M. Gross. Particle-based fluid-fluid interaction. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 237–244, 2005.

[MST+04] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids. *Computer Animation and Virtual Worlds*, 15(34):159–171, 2004.

[MST10] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74, 2010.

[Nex11] Next Limit Technologies. Realflow 2012, Hybrido. White Paper, 2011.

[NGL10] R. Narain, A. Golas, and M. C. Lin. Free-flowing granular materials with two-way solid coupling. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 29(6):173:1–173:10, 2010.

[NVI11] NVIDIA. mental ray (version 3.9). *http://www.mentalimages.com*, 2011.

[OK12] J. Orthmann and A. Kolb. Temporal blending for adaptive sph. *Computer Graphics Forum*, 31(8):2436–2449, 2012.

[Ope05] OpenMP Architecture Review Board. *OpenMP Application Program Interface, Version 2.5*, May 2005.

[OS88] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[Ov08] J. Onderik and R. Ďurikovič. Efficient Neighbor Search for Particle-based Fluids. *Journal of the Applied Mathematics, Statistics and Informatics (JAMSI)*, 4(1):29–43, 2008.

[PAKF13] S. Patkar, M. Aanjaneya, D. Karpman, and R. Fedkiw. A Hybrid Lagrangian-Eulerian Formulation for Bubble Generation and Dynamics. In *Proceedings Eurographics/ACM Siggraph Symposium on Computer Animation*, 2013.

[Pan04] A. Panizzio. *Physical and numerical modelling of subaerial landslide generated waves*. PhD thesis, Universita degli studi di L'Aquila, 2004.

[PDC+03] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50, 2003.

[Pea86] D. Peachey. Modeling waves and surf. In *Proceedings of SIGGRAPH*, pages 65–74, 1986.

[PF01] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 2–2, 2001.

[PTB+03] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle-Based Simulation of Fluids. *Computer Graphics Forum (Proceedings of Eurographics)*, 22:401–410, 2003.

[RWT11] K. Raveendran, C. Wojtan, and G. Turk. Hybrid Smoothed Particle Hydrodynamics. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 33–42, 2011.

[SB12] H. Schechter and R. Bridson. Ghost SPH for animating water. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 31(4):61:1–61:8, 2012.

[Sch02] H. Schmidl. *Optimization-Based-Animation*. PhD thesis, University of Miami, 2002.

[SF95] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of SIGGRAPH*, pages 129–136, 1995.

[SG11] B. Solenthaler and M. Gross. Two-Scale Particle Simulation. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 30(4):72:1–72:8, 2011.

[Sir99] W. A. Sirignano. *Fluid dynamics and transport of droplets and sprays*. Cambridge University Press, 1999.

[SL03] S. Shao and Y. Lo. Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface. *Advances in water resources*, 26(7):787–800, 2003.

[SOH99] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating Sand, Mud, and Snow. *Computer Graphics Forum*, 28:1–11, 1999.

[SP08] B. Solenthaler and R. Pajarola. Density Contrast SPH Interfaces. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–218, 2008.

[SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 28:40:1–40:6, 2009.

[Spr05] V. Springel. The cosmological simulation code GADGET-2. *Mon. Not. R. Astron. Soc.*, 364:1105–1134, 2005.

[SRF05] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 24:910–914, 2005.

[SSP07] B. Solenthaler, J. Schläfli, and R. Pajarola. A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82, 2007.

[Sta99] J. Stam. Stable fluids. *Proceedings of Computer graphics and interactive techniques*, pages 121–128, 1999.

[SU94] J. Steinhoff and D. Underhill. Modification of the euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6:2738–2744, 1994.

[SWB+06] I. Sbalzarini, J. Walther, M. Bergdorf, S. Hieber, E. Kosalis, and P. Koumoutsakos. PPM - A highly efficient parallel particle-mesh library for the simulation of continuum systems. *Journal of Computational Physics*, 215(2):566–588, 2006.

[Tes04] J. Tessendorf. Simulating Ocean Water. In *SIGGRAPH Course Notes*. ACM, 1999-2004.

[TFK+03] 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*, 22(3):391–400, 2003.

[THM+03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of Vision, Modeling, Visualization (VMV)*, pages 47–54, 2003.

[TKPR06] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–13, 2006.

[TMFSG07] N. Thürey, M. Müller-Fischer, S. Schirm, and M. Gross. Real-time Breaking Waves for Shallow Water Simulations. *Proceedings of the Pacific Conference on Computer Graphics and Applications*, pages 39–46, 2007.

[TRS06] N. Thürey, U. Rüde, and M. Stamminger. Animation of Open Water Phenomena with Coupled Shallow Water and Free Surface Simulations. *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 157–164, 2006.

[TSS+07] N. Thürey, 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 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 191–198, 2007.

[VCC98] B. Vermuri, Y. Cao, and L. Chen. Fast collision detection algorithms with applications to particle flow. *Computer Graphics Forum*, 17(2):121–134, 1998.

[Ver67] L. Verlet. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical Review*, 159(1):98–103, 1967.

[WS95] M. Warren and J. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87(1-2):266–290, 1995.

[ZB05] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Transactions on Graphics (Proceedings SIGGRAPH)*, 24:965–972, 2005.

[ZSP08] Y. Zhang, B. Solenthaler, and R. Pajarola. Adaptive Sampling and Rendering of Fluids on the GPU. In *Proceedings Symposium on Point-Based Graphics*, pages 137–146, 2008.

[ZYP06] W. Zheng, J.-H. Yong, and J.-C. Paul. Simulation of bubbles. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 325–333, 2006.