UNIVERSITÄT SIEGEN

Spatially Adaptive Smoothed Particle Hydrodynamics

DISSERTATION
zur Erlangung des Grades eines Doktors der Naturwissenschaften

vorgelegt von
M. Sc. Rene Winchenbach

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

Siegen 2021

Betreuer und erster Gutachter
Prof. Dr. Andreas Kolb
Universität Siegen

Externer Gutachter
Prof. Dr. Nils Thuerey
Technische Universität München

Tag der mündlichen Prüfung
16. Februar 2022

Zusammenfassung

Heutzutage ist die Strömungsmechanik ein wichtiges Forschungsgebiet in vielen Bereichen, z. B. bei Spezialeffekten in Filmen, bei der Planung von Küstenstrukturen im Ingenieurwesen oder bei der Simulation astrophysikalischer Phänomene. Obwohl jedes dieser Probleme sehr unterschiedliche Szenarien beinhaltet, verwenden sie alle dasselbe zugrunde liegende physikalische Modell für Flüssigkeitsströmungen, und daher sind auch die Simulationsansätze, die zur Simulation des Modells in jedem Problem verwendet werden, identisch. Ein häufig verwendeter Ansatz ist die Methode der Smoothed Particle Hydrodynamics (SPH), ein Lagrangescher Ansatz zur Lösung der Navier-Stokes-Gleichungen.

Idealerweise sollte die Auflösung der Simulation so hoch wie möglich sein, jedoch sind globale Auflösungserhöhungen nicht praktikabel, während vorherige Methoden zur lokalen Auflösungsänderung nicht praktisch nutzbar waren. In dieser Dissertation werden Forschungsarbeiten zu räumlich adaptiven SPH Methoden vorgestellt, die sich mit diesen Einschränkungen befassen, inklusive grundlegender und praktischer Probleme. Die vorgestellten Methdeon erlauben Simulationen mit um Größenordnungen höheren Adaptivitäten als in früheren Ansätzen.

Der Schwerpunkt der Forschung in dieser Dissertation liegt auf dem Prozess der Aufteilung eines einzelnen Partikels mit niedriger Auflösung in mehrere Partikel mit höherer Auflösung. Durch die Einführung eines kontinuierlichen adaptiven Prozesses mit einem neuartigen zeitlichen Blending und einem neuen Konzept der Umverteilung der Auflösung, wurden Simulationen mit einer um vier Größenordnungen höheren Adaptivität als bisher möglich. Diese Prozesse wurden weiter verbessert indem eine neuartige Optimierungsstrategie für die Verfeinerung verwendet wurde, welche a priori und auch während der Simulation selbst durchgeführt werden kann sowie ein verbessertes zeitliches Blending. Die verbesserten Prozesse stabilisieren die räumlich adaptiven Simulationen erheblich und ermöglichen praktischere und zuverlässigere Simulationen.

Ein weiterer wichtiger Schwerpunkt der vorgestellten Forschungsarbeiten ist die Behandlung von Rändern bei räumlich adaptiven Simulationen. Unter der Annahme dass die lokalen Grenzen planar sind, wurde eine analytische Lösung für planare Grenzgeometrien verwendet um einen skaleninvarianten Ansatz für die Behandlung von Grenzen zu entwickeln, der für räumlich adaptive Simulationen verwendet werden kann. Die Randbehandlung für nicht-adaptive Simulationen wurde durch diesen Ansatz ebenfalls erheblich verbessert, da dieser genauere Interaktionen mit Randmerkmalen unterhalb der Partikelauflösung ermöglicht.

Zuletzt werden Verfahren zu GPU-basierten Beschleunigungsstrukturen und Algorithmen erforscht, die die effiziente Implementierung und das Rendern von räumlich adaptiven SPH Simulation ermöglichen. Außerdem werden Methoden für anisotropes Rendering und zur Reduktion vom Speicherverbrauch vorgestellt.

Abstract

Fluid mechanics is an important area of research in many fields, e.g., special effects in movies, coastal structure design in engineering or the simulation of astrophysical phenomena. While each of these problems involve very different scenarios, they all use the same underlying physical model for fluid flows, and the simulation approaches used to simulate the fluid in each problem are also the same. One commonly used approach is the Smoothed Particle Hydrodynamics method, which is a Lagrangian approach to solving the Navier-Stokes equations.

To obtain better computational performance and more detailed simulations, the achievable resolution of the simulation should be made as high as possible. Uniform, global increases in resolution are however computationally prohibitive, whereas existing methods utilizing local changes in resolution suffer from various issues. This dissertation presents research on spatially adaptive Smoothed Particle Hydrodynamics to address many of these issues, including fundamental problems, e.g., simulation stability, and practical problems, e.g., computational performance. The findings presented in this dissertation allow simulations to have adaptivity orders of magnitudes higher than that in prior work.

The primary focus of the research in this dissertation is on the process of splitting a single particle with low resolution into many particles with higher resolution. By introducing a continuous adaptive process with a novel temporal blending process and a new concept of resolution sharing, simulations were shown to have adaptivity three orders of magnitude higher than what was previously possible. These processes were then further improved using a novel optimization strategy for refinement that can be carried out a priori and also during the simulation itself, with the latter utilizing evolutionary optimization, as well as an enhanced temporal blending scheme. The improved methods significantly stabilize spatially adaptive simulations, allowing for more practical and reliable simulations.

Another important focus of the presented research is on boundary handling for spatially adaptive simulations. By assuming the local boundaries are flat, an analytic solution for flat boundary geometries was used to develop a scale-invariant boundary handling approach that can readily be utilized for spatially adaptive simulations. Boundary handling for non-adaptive simulations was also significantly improved by this approach as it allows for more accurate interactions with boundary features below particle resolution.

Finally, this dissertation also covers research on GPU-based acceleration structures and algorithms that enable the efficient implementation and on-the-fly rendering of spatially adaptive fluids. Mechanisms to both handle anisotropic Smoothed Particle Hydrodynamics simulations and to significantly reduce the memory usage of both adaptive and non-adaptive simulations are also presented.

Acknowledgements

While a dissertation is the shiny cumulation of many years of research and, hopefully, a comprehensive scientific document, this page is dedicated to the countless people involved behind the curtains that kept the research moving forward. While there is much practice over the years in writing scientific work, writing a heartfelt acknowledgement is rarely practiced even though it might be the most important part of a cumulative dissertation as the scientific work has already been published and peer reviewed and the dissertation text is only connecting the dots, whereas this part is rarely mentioned and neither can it be peer reviewed and cited.

The financial support of the Deutsche Forschungsgemeinschaft (DFG Grant Ko-2960-15/1) is greatly acknowledged and appreciated.

First and foremost, I would like to thank my supervisor Prof. Dr. Andreas Kolb for his support of my research on a professional and personal level. While we may have had our differences, we always found a way to reconcile our ideas and approaches. I am deeply appreciative of all the time and feedback he provided over all these years, for him allowing me to follow my own ideas freely and providing me with this opportunity that would otherwise not have been possible.

I would also like to thank my predecessor Dr. Hendrik Hochstetter, who served as my superior when I started working at the CG group at the University of Siegen as a student. His support of my initial ideas, encouragement and assistance in bringing these ideas into a publication at SCA '16 were an important step on the path towards this dissertation and without his support I would have never been able to go into this direction.

I would also like to thank all the current and former members of the CG and CV groups at the University of Siegen for being an important part of the environment that enabled this dissertation to come to be. At this point, I would also like to thank Prof. Dr. Michael Möller of the CV group for his past and continued assistance, especially with research topics beyond the scope of this dissertation's topic.

Outside of the University of Siegen, I would also like to acknowledge the many researchers and professionals that I had the great pleasure of interacting with at various conferences and for their feedback, inspiration and motivation. I would also like to offer my deep gratitude towards the anonymous reviewers that provided valuable and fair criticism towards all of the research published in this dissertation.

I would also like to thank my parents Heidemarie and Wolfgang, and the rest of my family, for providing me with the opportunity and support on this long and bumpy road. It was not always easy and it did not always go as planned but I always felt comfort in knowing that I had a place to call home and that I was supported.

Finally, I would like to thank all of my friends for putting up with me over this long journey and for keeping me sane. Without their support this dissertation would not have been possible and I am sincerely thankful for each and every one of you.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction and Foundations

Fluid simulations have become an important part of Computer Animation in recent times driven by a desire for more detailed and realistic simulations, especially for situations that are impossible to create using actual fluids, either due to their impracticality or potential safety concerns. Many different simulation approaches exist within Computer Animation to achieve these simulations; where this dissertation focuses on the Lagrangian Smoothed Particle Hydrodynamics method, as it is versatile, physically motivated and computationally efficient [Kos+19]. However, using high global resolutions is difficult as computational requirements scale cubically with reductions of the particle radius. Locally adaptive resolutions focus computational resources where they are important through local changes in resolution and can significantly reduce computational requirements [SG11; OK12].

Within Lagrangian simulation approaches, such as SPH, changing the resolution locally while ensuring consistent and computationally efficient simulations across varying resolutions is a challenging problem. Many different spatially adaptive methods have been proposed for SPH before the start of this research project, e.g., Adams et al. [Ada+07], Solenthaler and Gross [SG11] and Orthmann and Kolb [OK12]. However, none of these methods are able to handle incompressible simulations at high adaptive ratios (e.g., of 1000 and higher), limiting their practical applicability. Other significant challenges faced in these methods include handling adaptive particle data without incurring significant computational overhead and handling boundary objects consistently across varying resolutions.

To address these issues, this dissertation first provides a general overview of the field of Smoothed Particle Hydrodynamics, its spatially adaptive variants and relevant mathematical foundations. This initial in-depth discussion is necessary as there exists a wide range of applied terminology and notations, and serves as a foundational framework for the reproduced papers. This overview also discusses how the publications reproduced within this dissertation cumulatively address the aforementioned issues. Moreover, some general concepts that are relevant to the papers are discussed, e.g., additional aspects regarding stability and accuracy. After this overview, the published papers are reproduced with added contextualization to position their arguments within the dissertation.

This cummulative dissertation contains six published works [WHK16; WHK17; WK19; WAK20; WK20; WK21] that cover adaptive simulation techniques, data handling, rendering and boundary handling for SPH. This work covers the corpus of work published as a student assistant and PhD student at the University of Siegen.

1.1 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) was initially conceived by Gingold and Monaghan [GM77] as a particle-based approach to solve hydrodynamic problems in astrophysical simulations of non-spherical stars. The method relies on concepts based on prior work by Bartlett [Bar63], Parzen [Par62] and Boneva, Kendall and Stepanov [BKS71]. This approach has been adapted to many other applications, including the simulation of incompressible liquids in Computer Animation [Kos+19]. The goal of this section is therefore to build a tenable foundation for the remainder of the dissertation by providing a derivation of the overall SPH method using prior work and a clear definition of important formulae.

In general, SPH works by discretizing continuous fields using Lagrangian particles that carry discrete quantities and a local interpolation scheme, and thus reconstructing the underlying continuous fields using spatially compact smoothing kernels. SPH is fundamentally built on an identity transformation of a field $A(\mathbf{x})$, which is defined at all points $\mathbf{x} \in \mathbb{R}^d$ in space by

$$A(\mathbf{x}) = \int_{\mathbb{R}^d} A(\mathbf{x}')\delta(\mathbf{x}-\mathbf{x}')d\mathbf{x}', \tag{1.1}$$

where δ is the Dirac delta function and $\mathbb{R}^d$ describes an infinite d-dimensional simulation domain. This identity is then modified by replacing δ with a compact smoothing kernel which converges to a delta function for vanishing size, i.e., $\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}',h) = \delta(\mathbf{x}-\mathbf{x}')$ and yields an approximation

$$A(\mathbf{x}) \approx \langle A(\mathbf{x})\rangle = \int_{\Omega} A(\mathbf{x}')W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}', \tag{1.2}$$

where h is the support radius, describing the size of the compact spherical support domain Ω, and W is a compact smoothing kernel. Note that the specific requirements for a kernel function will be discussed in further detail in Section 1.2. The chevrons $\langle A(\mathbf{x})\rangle$ indicate that this is a pointwise interpolation of $A(\mathbf{x})$. For readability, the chevrons will be discarded if it can easily be inferred that the quantity referred to is an approximation. The next step is then to introduce the density $\rho(\mathbf{x})$ at a given spatial position $\mathbf{x}$ into the integral, which yields

$$\langle A(\mathbf{x})\rangle = \int_{\Omega} \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')}W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}'. \tag{1.3}$$

This approximation is then discretized by replacing the integral with a summation and the mass element $\rho\, dV$ with the particle mass m, yielding

$$\langle A(\mathbf{x})\rangle \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W(\mathbf{x}-\mathbf{x}_j,h), \tag{1.4}$$

with $\mathcal{N}_\mathbf{x}$ being the set of neighbors relative to $\mathbf{x}$, i.e., the set of all particles with $|\mathbf{x}-\mathbf{x}_j| \leq h$, and ρ, which can be evaluated using eq. 1.3 as

$$\rho(\mathbf{x}) = \sum_{j\in\mathcal{N}_\mathbf{x}} m_j W(\mathbf{x}-\mathbf{x}_j,h). \tag{1.5}$$

The numerical accuracy of this interpolation strongly depends on the local particle distribution, i.e., the so-called color field value that is calculated as

$$C(\mathbf{x}) = \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} W(\mathbf{x}-\mathbf{x}_j, h), \tag{1.6}$$

which ideally evaluates to 1 everywhere, and consequently the gradient of this field should be $\mathbf{0}$ everywhere. However, the particle ordering in actual simulations can become very disordered, which in turn limits the achievable numerical accuracy. The particle ordering can be restored, at least in weakly compressible formulations, using particle-shifting [Sun+19] where particles are shifted locally to yield a more isotropic particle distribution. It is important to note that, at the time of this dissertation, only some initial research has been proposed regarding particle-shifting methods that are applicable to incompressible SPH simulations [KGS19]. Moreover, no particle-shifting methods so far are applicable to spatially adaptive simulations, where the non-isotropic particle distribution on resolution interfaces is still an open numerical problem, for further details see Chapter 3.

In addition to the reconstruction of the actual underlying field $A(\mathbf{x})$, gradient terms play an important role in fluid dynamics. The gradient of a field $A(\mathbf{x})$ can be derived straightforwardly by applying the gradient operator $\frac{\partial}{\partial \mathbf{x}}$ to eq. 1.4, giving

$$\langle \nabla_\mathbf{x} A(\mathbf{x})\rangle \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j \nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}_j, h), \tag{1.7}$$

where $\nabla_\mathbf{x}$ denotes the gradient taken with respect to the position $\mathbf{x}$. Note that the subscript on the gradient operator is generally dropped for readability when the respective value is clear from context. In addition to scalar fields, vector-valued fields can be handled by replacing the scalar field $A(\mathbf{x})$ in eqs. 1.4 and 1.7 with a vector-valued field $\mathbf{A}(\mathbf{x})$, which yields the following set of general SPH interpolants for vector fields

$$\begin{aligned}
\langle \mathbf{A}(\mathbf{x})\rangle &\approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} \mathbf{A}_j W(\mathbf{x}-\mathbf{x}_j, h),\\
\langle \nabla_\mathbf{x}\cdot \mathbf{A}(\mathbf{x})\rangle &\approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} \mathbf{A}_j \cdot \nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}_j, h),\\
\langle \nabla_\mathbf{x}\times \mathbf{A}(\mathbf{x})\rangle &\approx -\sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} \mathbf{A}_j \times \nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}_j, h).
\end{aligned} \tag{1.8}$$

However, these gradient expressions yield poor gradient estimates as these formulations do not correctly handle constant fields, i.e., $\nabla_\mathbf{x} A(\mathbf{x}) \neq 0$ for $A(\mathbf{x}) = c : \forall \mathbf{x} \in \mathbb{R}^d$ and $c \in \mathbb{R}$. Moreover, to abide by Newton's Third Law [New87], these would need to be symmetric with respect to the interaction between two particles i and j, i.e., $F_{i\to j} = -F_{j\to i}$, but this formulation does not ensure this property. By taking the resulting error terms of eq. 1.7, into consideration [Pri12], gradient formulations that resolve these problems can be derived. From [Pri12], a formulation that is exact for constant functions is defined as

$$\begin{aligned}
\nabla_\mathbf{x} A(\mathbf{x}) &\approx \frac{1}{\rho(\mathbf{x})}\left[\langle \nabla_\mathbf{x}(\rho(\mathbf{x})A(\mathbf{x}))\rangle - A(\mathbf{x})\langle \nabla_\mathbf{x}\rho(\mathbf{x})\rangle\right]\\
&= \frac{1}{\rho(\mathbf{x})}\sum_{j\in\mathcal{N}} m_j (A_j - A(\mathbf{x}))\nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}_j, h),
\end{aligned} \tag{1.9}$$

while a formulation which ensures symmetric gradients is defined as

(1.10)
$$\begin{aligned}\nabla_{\mathbf{x}}A(\mathbf{x}) &\approx \rho(\mathbf{x})\left[\frac{A(\mathbf{x})}{\rho(\mathbf{x})^2}\langle\nabla_{\mathbf{x}}\rho(\mathbf{x})\rangle + \left\langle\nabla_{\mathbf{x}}\frac{A(\mathbf{x})}{\rho(\mathbf{x})}\right\rangle\right]\\ &= \rho(\mathbf{x})\sum_{j\in\mathcal{N}} m_j\left(\frac{A_j}{\rho_j^2}+\frac{A(\mathbf{x})}{\rho(\mathbf{x})^2}\right)\nabla_{\mathbf{x}}W(\mathbf{x}-\mathbf{x}_j,h),\end{aligned}$$

with analogous terms for vector-valued fields $\mathbf{A}(\mathbf{x})$. Note that in addition to these gradient terms, another set of derivatives can appear if the support radius h is not constant, i.e., h varies over time, or depends on other properties of the particle. Consequently, when applying a derivative operator on eq. 1.4, the derivative of h by $\mathbf{x}$, i.e., $\frac{\partial h}{\partial \mathbf{x}}$, does not become 0 in all cases.

In Computer Animation contexts, this issue is generally avoided by setting the support radius to be constant for each particle and consequently removing any derivative of the support radius. Note that the papers reproduced in Chapters 2 and 3 utilize non-constant support radii and discuss the effects of doing so in more detail, especially in regards to simulations with symmetric SPH formulations.

In CFD contexts the derivative of the support radius is generally solved by relating the support radius to the density of a particle, i.e., $h \propto \rho$, where a corrective term $\frac{1}{\Omega_i}$ [Mon92; Pri12] is introduced for each particle i, where

(1.11)
$$\Omega_i = \left[1 + \frac{h_i}{\rho_i d}\sum_{j\in\mathcal{N}_i} m_j \frac{\partial W(\mathbf{x}_i - \mathbf{x}_j, h)}{\partial h}\right],$$

with d being the dimensionality of the simulation. This term is then multiplied onto all gradient estimates, e.g., onto the basic gradient estimate in eq. 1.7:

(1.12)
$$\nabla_i A_i \approx \frac{1}{\Omega_i}\sum_j \frac{m_j}{\rho_j} A_j \nabla_i W_{ij},$$

where ∇_i indicates a spatial derivative with respect to the position of particle i.

At this point it is important to note that from an underlying mathematical standpoint, SPH can readily handle varying particle sizes, i.e., particles of different volume. However, issues can arise due to numerical considerations and changes in spatial resolution, which will be discussed later on in Sec. 1.6.

Furthermore, handling particles of different phases, i.e., multi-phase flows, can require significant adjustments to the SPH model presented here [SP08; Yan+16; RXL21], and handling spatial adaptivity with multi-phase flows is still an open problem. Moreover, significant research regarding the underlying accuracy and stability of SPH is still ongoing, e.g., with regards to particle isotropy and kernel functions; see for example [DA12; Vac+21]. While these issues are important to SPH and for future research, they are outside of the scope of this research and will not be further discussed in this dissertation.

1.2 Kernel functions

So far only an arbitrary kernel function W was used; however, kernel functions have to fulfill several properties in order to be numerically reasonable [Pri12]:

- W has to be positive and monotonically decreasing with increasing distance between particles, i.e., $W(\mathbf{x},h) \geq 0 : \forall \mathbf{x} \in \mathbb{R}^n$ and $W(\mathbf{x},h) \geq W(\mathbf{x}',h) : \forall |\mathbf{x}'| \geq |\mathbf{x}|, \mathbf{x}' \in \mathbb{R}^n, \mathbf{x} \in \mathbb{R}^n$,
- symmetric with respect to $\mathbf{x}-\mathbf{x}'$, i.e., $W(\mathbf{x}-\mathbf{x}',h) = W(\mathbf{x}'-\mathbf{x},h)$,
- normalized, i.e., $\int_\Omega W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}' = 1$, and
- converges towards a Dirac delta function with decreasing support radius, i.e., $\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}',h) = \delta(\mathbf{x}-\mathbf{x}')$.

Gaussian smoothing kernels fulfill all these and are defined as [DA12]

$$W(\mathbf{x}-\mathbf{x}',h) = \frac{C_d}{h^d} e^{-\frac{||\mathbf{x}-\mathbf{x}'||^2}{h^2}}, \tag{1.13}$$

with the normalization constants $\frac{1}{\sqrt{\pi}}$, $\frac{1}{\pi}$ and $\frac{1}{\pi\sqrt{\pi}}$ in 1D, 2D and 3D, respectively. Even though the influence of particles farther away than h rapidly diminishes, Gaussian smoothing kernels are not practically useful as they have no compact support. Consequently, compact kernel functions are generally preferred as they limit the number of useful neighbors of a particle [DA12]. Note that here the *support* of the Gaussian kernel is often referred to as its *smoothing scale* and the support radius H, chosen as a multiple of the smoothing scale, is denoted as the cutoff distance after which particles are no longer considered. Accordingly, there exists a ratio $\frac{H}{h}$ between the smoothing scale h and the support radius H. Within Computer Animation, Gaussian kernels find no practical application so this distinction is generally dropped by implicitly assuming that $\frac{H}{h} = 1$ and, consequently, $H = h$, which simplifies many equations and notations, without loss of generality. A generic kernel function and its spatial derivative can be defined for the interaction of two points $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{x}' \in \mathbb{R}^d$ as [DA12]

$$W(\mathbf{x}-\mathbf{x}',h) = \frac{C_d}{h^d}\hat{W}\left(\frac{|\mathbf{x}-\mathbf{x}'|}{h}\right), \nabla_{\mathbf{x}} W(\mathbf{x}-\mathbf{x}',h) = \frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|}\frac{C_d}{h^{d+1}}\frac{\partial}{\partial\frac{|r|}{h}}\hat{W}\left(\frac{|r|}{h}\right), \tag{1.14}$$

with $\hat{W}$ being the actual kernel with a compact support of 1. Many different choices exist for this kernel; in this dissertation the most used ones are the Wendland4 kernel and the cubic spline kernel. The cubic spline kernel is given by [DA12]

$$\hat{W}(q) = [1-q]_+^3 - 4\left[\frac{1}{2}-q\right]_+^3, \tag{1.15}$$

where $[\cdot]_+ = \max(0,\cdot)$, with normalization constants $\left\{\frac{8}{3}, \frac{80}{7\pi}, \frac{16}{\pi}\right\}$ in 1D, 2D and 3D. On the other hand, the Wendland4 kernel is given by [DA12]

$$\hat{W}(q) = [1-q]_+^6\left[1+6q+\frac{35}{3}q^2\right], \tag{1.16}$$

with normalization constants $\left\{\frac{9}{\pi}, \frac{495}{32\pi}\right\}$ in 2D and 3D, respectively. Based on the numerical properties of the kernel, see the extensive discussions by Dehnen and Aly [DA12], each kernel function also has an ideal number of neighbors N_h, which is 55 for the cubic spline kernel and 200 for the Wendland4 kernel. However, in Computer Animation the number of neighbors is often constrained by computational requirements and generally set to be lower, i.e., the Wendland4 kernel function is used with a desired number of neighbors of around 60. Using this desired number of neighbors, it is straightforward to determine the support radius of a particle in 3D, see Chapter 2, as

$$h_i = \sqrt[3]{N_h \frac{3}{4\pi}}\sqrt[3]{V_i}, \tag{1.17}$$

with V_i being either the rest volume of a particle in a Computer Animation context, i.e., $V_i = \frac{m_i}{\rho_{i,0}}$, or the apparent volume of a particle in a CFD context, i.e., $V_i = \frac{m_i}{\rho_i}$. Note that the latter introduces the dependency $h \propto \rho$, which means that derivatives of the support radius h are non-zero, e.g., $\frac{dh}{dt} \neq 0$, which necessitates the derivative terms discussed at the end of the prior section. Furthermore, this dependence also introduces a coupled problem as the support radius impacts the density estimate used for calculating it, which is usually resolved through an iterative solving process [Pri12]. The paper reprinted in Chapter 2 discusses alternative ways to control the support radius to improve computational performance and to reduce memory consumption. It is important to note that numerical evaluations of kernel functions, e.g., pairing and tensile instabilities as well as ideal neighborhood sizes and distributions, have only been thoroughly performed for uniform resolution simulations [DA12], but not for for spatially adaptive simulations.

So far, only an arbitrary support radius h has been used for all calculations involving kernel functions, i.e., h is constant and equal for all particles, but this is generally not the case, e.g., in spatially adaptive simulations. In practice, three different formulations exist to determine the support radius used when two particles i and j are interacting: (i) the scatter-based formulation $h_{ij} = h_j$, (ii) the gather-based formulation $h_{ij} = h_i$, and (iii) the symmetric formulation $h_{ij} = \frac{h_i+h_j}{2}$. In a CFD context the scatter-based formulation has found wide adoption as this formulation reduces the complexity of support radius derivative terms, whereas in a Computer Animation context the symmetric formulation is used for spatially adaptive simulations due to different underlying pressure solvers, see Chapters 3 and 7. Using the support radius of an interaction h_{ij} between two particles i and j, the basic SPH interpolant (1.4) can re-expressed as

$$A_i \approx \sum_{j\in\mathcal{N}_i} \frac{m_j}{\rho_j} A_j W_{ij}, \tag{1.18}$$

where $\mathcal{N}_i$ is the set of neighbors of particle i and $W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h_{ij})$. Finally, instead of using spherical support domains described by a single scalar support radius h, an alternative formulation of SPH utilizes ellipsoidal-shaped support domains [Owe+98], i.e., the support *radius* becomes a matrix G such that $G = \frac{1}{h} I_d$ results in $W(\mathbf{r}, h) = W(\mathbf{r}, G)$. Consequently, the kernel function is defined as

$$W(\mathbf{r}, G) = \frac{C_d}{\det(G)} \hat{W}(|G\mathbf{r}|), \tag{1.19}$$

where $\det(G)$ is the determinant of G. These anisotropic SPH formulations have found no significant adoption in recent SPH simulations within the Computer Animation community, due to their increased complexity offering few numerical advantages for uniform simulations [Owe+98]. Despite this, they have found wide usage in rendering contexts as the anisotropic support domain can match free surface regions better [YT13], see Chapter 5. Furthermore, determining the ideal support domain size for an anisotropic simulation is still an open problem, especially for spatially adaptive simulation approaches. For a more in-depth discussion of anisotropic SPH formulations see the paper reproduced in Chapter 5.

1.3 Governing equations

The foundations of SPH established in Sections 1.1 and 1.2 build a mathematical framework that can be used for fluid simulations. Implementing the simulation would however require a compatible set of equations describing the fluid mechanics of the system to be simulated. Hence this section describes the derivation of this set of equations, and also how they are discretized using SPH. The basic underlying equation to most SPH models [Pri12; Kos+19] is the continuity equation which relates the rate of change of density over time to the divergence of the flow, and is given as

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot\mathbf{v}, \tag{1.20}$$

with $\frac{D}{Dt}$ being the material derivative. Note that for an incompressible liquid, i.e., ρ is constant, the rate of change of density is 0, which also implies that fluid is divergence-free. This continuity equation is then used in conjunction with the incompressible Navier-Stokes equation, which is defined as

$$\rho\frac{D\mathbf{v}}{Dt} = -\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{f}^{\text{ext}}, \tag{1.21}$$

where $\mu\nabla^2\mathbf{v}$ describes the viscosity of the flow, $\mathbf{f}^{\text{ext}}$ are external body forces applied on the system, e.g., gravity, and ∇p is the pressure force.

In Computer Animation the most commonly used time integration scheme is the semi-implicit Euler scheme [Ihm+13; BK15], i.e., $\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + \Delta t\frac{D\mathbf{v}^t}{Dt}$, where the timestep Δt plays an important role to the simulation stability. When solving the Navier-Stokes equation (1.21), which is a partial differential equation, the Courant-Friedrichs-Lewy (CFL) condition provides a necessary condition for convergence of the solution and provides an upper bound on the global timestep Δt as

$$\Delta t \leq \lambda\frac{h_{\text{min}}}{|\mathbf{v}_{\text{max}}|}, \tag{1.22}$$

where h_{min} is the smallest support radius across all particles, $\mathbf{v}_{\text{max}}$ is the largest velocity magnitude of any particle and $\lambda \leq 1$ is the CFL scaling factor. Intuitively, any choice of $\lambda \leq 1$ would be sufficient as this prevents particles from moving more than their support radius per timestep, but this is generally not sufficient to ensure a stable simulation. In practice, many other numerical effects need to be taken into account, e.g., how particles can move in and out of the support domain of other particles and, due to the compact kernel function, cause errors in gradient estimates, which will be discussed in Sec. 1.6. Furthermore, a timestep with $\lambda \geq \frac{1}{2}$ is not sufficient in ensuring that particles do not move through each other, i.e., two particles moving directly towards each other can still be outside of each other's

ALGORITHM 1.1: A simple schematic SPH simulation algorithm.

```
for all particles $i$:
    Compute $\mathcal{N}_i = \{j| \|\mathbf{x}_i - \mathbf{x}_j\| \leq h_{ij}\}$
for all particles $i$:
    Compute $\rho_i = \sum_{j\in\mathcal{N}_i} m_j W_{ij}$
for all particles $i$:
    Compute $\mathbf{v}_i^{\text{adv}} = \mathbf{v}_i + \Delta t \frac{1}{\rho_i}\left[\mu\nabla^2\mathbf{v}_i + \mathbf{f}^{\text{ext}_i}\right]$
for all particles $i$:
    Solve for $\nabla p$ such that $\frac{D\rho}{Dt} = 0$
for all particles $i$:
    Integrate $\mathbf{v}' = \mathbf{v}^{\text{adv}} + \frac{\Delta t}{\rho}\nabla p, \mathbf{x}' = \mathbf{x} + \Delta t\mathbf{v}'$
for all particles $i$:
    Update $\Delta t = \lambda\frac{h_{\text{max}}}{|\mathbf{v}_{\text{max}}|}$
```

support radius at one timestep but pass through each other in the next. Consequently, a common heuristic choice for λ is 0.4 [Mon92; Ihm+13], as this provides a reasonable balance between stability and computational expediency.

Solving the incompressible Navier-Stokes equation (1.21) can be split into a multi-step process [Ihm+13], where (i) for a given an initial velocity field $\mathbf{v}$ the advected velocity field $\mathbf{v}^{\text{adv}}$ is evaluated as

$$\mathbf{v}^{\text{adv}} = \mathbf{v} + \Delta t\frac{1}{\rho}\left[\mu\nabla^2\mathbf{v} + \mathbf{f}^{\text{ext}}\right], \tag{1.23}$$

(ii) a pressure field p is then determined such that $\frac{D\rho}{Dt} = 0$ to enforce the continuity equation, by solving for pressure values on individual particles i using pressure forces defined as

$$\nabla p_i = \sum_{j\in\mathcal{N}_i} m_j\left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right)\nabla_i W_{ij}, \tag{1.24}$$

and (iii) a new velocity field $\mathbf{v}'$ is calculated using $\mathbf{v}^{\text{adv}}$ and ∇p as

$$\mathbf{v}' = \mathbf{v}^{\text{adv}} + \Delta t\frac{1}{\rho}\nabla p. \tag{1.25}$$

The main challenge in this process is determining the pressure force that enforces the continuity equation, and multiple approaches exist to solve this problem [Ihm+13; BK15]. For more details on this derivation, see the Eurographics SPH Tutorial by Koschier et al. [Kos+19]. Assuming that neighborlists are used in the simulation, a simple simulation algorithm can then be derived, see Algorithm 1.1. Note that additional forces, e.g., due to vorticity refinement [Ben+18] or surface tension [AAT13] need to be included in the calculation of the advected velocity.

The most commonly used approaches in Computer Animation to solve for the pressure forces first involve using the continuity equation (1.20) to predict the density in the next timestep [Ihm+13; BK15]. Replacing the material derivative $\frac{D\rho}{Dt}$ in the continuity equation with a finite forwards difference, $\frac{D\rho}{Dt} = \frac{\rho^{t+\Delta t} - \rho^t}{\Delta t}$, and using a difference formulation,$\nabla\cdot\mathbf{v}(\mathbf{x}) = \rho(\mathbf{x})\sum_{j\in\mathcal{N}_\mathbf{x}} m_j(\mathbf{v}_j - \mathbf{v}(\mathbf{x}))\cdot\nabla_\mathbf{x} W(\mathbf{x} - \mathbf{x}_j, h)$, for the velocity divergence yields [Ihm+13]

$$\frac{\rho_i^{t+\Delta t} - \rho_i^t}{\Delta t} = \sum_{j \in \mathcal{N}_i} m_j \mathbf{v}_{ij}^{t+\Delta t} \nabla_i W_{ij}, \tag{1.26}$$

where $\mathbf{v}_{ij}^{t+\Delta t} = \mathbf{v}_i^{t+\Delta t} - \mathbf{v}_j^{t+\Delta t}$. Using a semi-implicit Euler scheme for the velocity, $\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + \Delta t \mathbf{a}^t$, eq. 1.26 can then further be refactored to yield

$$\rho_i^{t+\Delta t} = \rho_i^t + \Delta t \left[\sum_{j \in \mathcal{N}_i} m_j \left(\mathbf{v}_i^t + \Delta t \mathbf{a}_i^t - \mathbf{v}_j^t - \Delta t \mathbf{a}_j^t \right) \nabla_i W_{ij} \right]. \tag{1.27}$$

Splitting the acceleration $\mathbf{a}$ of a particle into the acceleration due to pressure $\mathbf{a}^p$ and the acceleration due to non-pressure forces $\mathbf{a}^{\text{adv}}$ gives

$$\begin{aligned} \rho_i^{t+\Delta t} = \rho_i + \Delta t & \left[\sum_{j \in \mathcal{N}_i} m_j \left(\mathbf{v}_i + \Delta t \mathbf{a}_i^{\text{adv}} - \mathbf{v}_j - \Delta t \mathbf{a}_j^{\text{adv}} \right) \nabla_i W_{ij} \right] \\ & + \Delta t^2 \left[\sum_{j \in \mathcal{N}_i} m_j \left(\mathbf{a}_i^p - \mathbf{a}_j^p \right) \nabla_i W_{ij} \right], \end{aligned} \tag{1.28}$$

where time indices t have been dropped. Eq. 1.28 can then be refactored into

$$\rho_i^{t+\Delta t} = \underbrace{\rho_i + \Delta t \left[\sum_{j \in \mathcal{N}_i} m_j \mathbf{v}_{ij}^{\text{adv}} \nabla_i W_{ij} \right]}_{\rho_i^{\text{adv}}} + \Delta t^2 \left[\sum_{j \in \mathcal{N}_i} m_j \mathbf{a}_{ij}^p \nabla_i W_{ij} \right]. \tag{1.29}$$

Finally, in order to solve for unknown accelerations due to pressure forces eq. 1.29 can be refactored into a system of equations

$$\Delta t^2 \left[\sum_{j \in \mathcal{N}_i} m_j \mathbf{a}_{ij}^p \nabla_i W_{ij} \right] = \rho_i^{t+\Delta t} - \rho_i^{\text{adv}}, \tag{1.30}$$

where the left-hand side depends on unknown pressure values p, and the right-hand side is also referred to as the *source term*. To solve this system of equations, $\rho_i^{t+\Delta t}$ needs to be determined. Setting $\rho_i^{t+\Delta t} = \rho_{0,i}$ results in a pressure force that will enforce an incompressible simulation, i.e., $\rho_i = \rho_{0,i}$, and setting $\rho_i^{t+\Delta t} = \rho_i$ will result in a pressure force that enforces a divergence-free simulation, i.e., $\frac{D\rho}{Dt} = 0$. Implicit Incompressible SPH (IISPH) [Ihm+13] only utilizes the incompressible variant, whereas Divergence-Free SPH (DFSPH) [BK15] first uses the divergence-free variant and then the incompressible variant to yield improved overall simulation behavior. Note that due to particle neighborhood deficiencies at free surfaces, solving eq. 1.30 directly can yield negative pressure values when enforcing incompressibility. This can be avoided by using $\max(\rho_{0,i} - \rho_i, 0)$ on the right-hand side of eq. 1.30.

Solving the system of equations 1.30 for pressure is generally done in an iterative process, where some stopping criterion needs to be chosen to decide when the solution is *good enough*. One straightforward solution is basing the stopping criterion on the rate of change of pressure per iteration. In practice [Ihm+13;

BK15], however, the stopping criterion is commonly based solely on the error in the solution, which can be done straightforwardly by evaluating

$$\epsilon_i = \rho_i^{t+\Delta t} - \rho_i^{\text{adv}} - \Delta t^2 \left[\sum_{j \in \mathcal{N}_i} m_j \mathbf{a}_{ij}^p \nabla_i W_{ij} \right], \tag{1.31}$$

where $\rho_i^{t+\Delta t}$ depends on the problem being solved. Using the rest density of each particle, the residual error ϵ can be utilized to determine an average relative error per particle as [Ihm+13]

$$\eta = \frac{\sum_{i=0}^{n-1} \frac{\epsilon_i}{\rho_{i,0}}}{n} \tag{1.32}$$

with n being the number of particles. However, this estimate does not yield desirable results in spatially adaptive simulations as the number of particles in regions with higher resolution will skew the average error, as these regions are usually close to the fluid surface with little compressive stress. Consequently, using a mass-weighed error term

$$\eta = \frac{\sum_{i=0}^{n-1} m_i \frac{\epsilon_i}{\rho_{i,0}}}{\sum_{i=0}^{n-1} m_i} \tag{1.33}$$

yields better convergence in practice. Regardless, finding the optimal convergence criteria for spatially adaptive simulations is still an open problem. Furthermore, the convergence of the iterative solver strongly depends on the timestep used [Ihm+13] where, larger timesteps generally require more iterations to converge. Accordingly, some optimal timestep Δt^{opt} exists such that it results in the overall lowest computational cost, i.e., a timestep that balances the number of steps per simulated time with a higher cost per individual step, but finding this optimal timestep is still an open problem.

Furthermore, larger timesteps yield increased errors in the estimation process, which will be discussed later in Sec. 1.6, and sudden changes in the timestep can yield significantly different convergence behaviors. This leads to issues in spatially adaptive simulations as particles being refined into higher resolutions reduce their support radii, which significantly reduces the largest feasible timestep satisfying the CFL condition. Accordingly, a sudden increase in resolution, as is the case during the initial refinement in a spatially adaptive simulation, results in a sudden change in the maximum permissible timestep. These errors can be diminished by limiting the change in the maximum permissible timestep before the occurrence of any significant change of resolution or scenarios that would cause a sudden change in timestep. However, it is generally not straightforward to detect these issues a priori, and addressing this problem is still an unsolved problem for black-box scenarios and beyond the scope of the research in this dissertation.

1.4 Boundary handling

Within most simulation scenarios, especially within Computer Animation, the fluid is generally not evaluated within an empty space, but is contained within some container or domain where other objects, e.g., rigid obstacles, influence the fluid flow behavior. In this regard, SPH simulations require both models for boundary conditions and models of boundary geometries, which are not generally represented in a Lagrangian form. In practice, many different boundary conditions exist, e.g., inlet and outlet conditions [Taf+17], open and periodic boundary conditions, as well as different varieties of rigid boundary conditions, e.g., no-slip and free-slip boundary conditions. In this dissertation only rigid boundary geometries and their integration into an SPH model are considered.

To integrate boundary geometries into the SPH model, the first step is to modify the local domain $\Omega_\mathbf{x}$ around a particle at position $\mathbf{x}$ based on its support radius h, as the simulation domain is no longer fully discretized using fluid particles. Consequently, if some boundary domain $\mathcal{B}$ overlaps with the local support domain $\Omega_\mathbf{x}$ then the discretization using fluid particles only applies in the non-overlapping region, i.e., $\Omega^f_\mathbf{x} = \Omega_\mathbf{x} \setminus \mathcal{B}$. The local boundary domain $\Omega^b_\mathbf{x} = \Omega_\mathbf{x} \cap \mathcal{B}$, however, cannot be directly discretized using fluid particles. Consequently, the continuous SPH interpolation operator eq. 1.2 becomes

$$A(\mathbf{x}) \approx \int_{\Omega^f_\mathbf{x}} \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}' + \int_{\Omega^b_\mathbf{x}} \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}', \tag{1.34}$$

where only the fluid contribution is discretized using particles, yielding

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij} + \int_{\Omega^b_\mathbf{x}} \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}'. \tag{1.35}$$

Accordingly, the main challenge in boundary handling is finding an appropriate way to discretize the boundary contributions to an SPH interpolant. In general, four categories of approaches exist in SPH to model boundary objects, which are:

1. External boundary handling methods
2. Particle-based methods
3. Wall-renormalization methods
4. Boundary-integral methods

Moreover, if pressure forces are calculated explicitly when interacting with boundaries, pressure values need to be evaluated for boundary domains, either pointwise or globally. Consequently, the different approaches will first be discussed, and then pressure extrapolation methods commonly used in Computer Animation will be outlined.

1.4.1 External boundary handling methods

These methods were initially utilized with weakly compressible SPH formulations and used a variety of approaches to model boundary effects. However, these methods were often not strongly physically motivated and difficult to integrate into SPH interpolants. Some examples include particle level sets [Los+08] and direct forcing [BTT09]. Owing to their non-physical nature, they are generally not integrated into the SPH interpolant, i.e., eq. 1.35 simply becomes

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij}, \tag{1.36}$$

which makes the physically accurate modelling of complex fluid-boundary effects impractical. Furthermore, implementing physically motivated and accurate two-way coupling effects in these methods is not readily possible.

1.4.2 Particle-based methods

Particle-based methods can be classified into three sub-categories: (i) full sampling methods, (ii) ghost-particle methods, and (iii) surface sampling methods.

Full sampling methods represent an entire boundary geometry with particles sampled as if the boundary geometry were part of the fluid domain [AHA12]. Accordingly, boundary particles are created as if they were fluid particles, and they can be directly integrated into an SPH simulation, albeit with a restricted integration scheme as they cannot move individually. Consequently, eq. 1.35 becomes

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij}, \tag{1.37}$$

where $\mathcal{N}_\mathbf{x}$ is the set of all neighboring particles, relative to $\mathbf{x}$, including boundary particles. However, sampling complex boundary object geometries completely with particles is a challenging problem. Some work has been done on generating optimal initial conditions in SPH [Die+15], which could be applied on this problem as well, but the additional number of particles required, relative to surface-only approaches, generally makes this approach unattractive.

Ghost-particle methods utilize virtual boundary particles that may not be sampled consistently across different fluid particles. These virtual particles are placed inside the boundary object or on its surface. This can be achieved in many ways, e.g., using local uniform stencils per particle [Fou+19], mirroring fluid particles into the boundary domain [YRS09], or by creating local optimized particle configurations [BGT17]. These particles can generally be integrated directly into an SPH model without further modification, i.e., eq. 1.35 becomes

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij}, \tag{1.38}$$

with $\mathcal{N}_\mathbf{x}$ is the set of all neighboring particles relative to $\mathbf{x}$, including boundary particles. However, locality can yield problems as different particles may have different boundary particle configurations [Fou+19], due to oversampling in sharp corners [YRS09] and/or be limited to mostly flat boundary regions [BGT17]. Accordingly, while these approaches can yield very desirable effects, they have also found limited application.

Surface particle methods utilize only a single layer of particles which is consistently sampled on the surface of the boundary object, where each particle carries an artificial volume to correct for the single layer and irregular sampling of the surface [Aki+12]. While this approach introduces some additional terms, i.e., artificial volumes and densities, it can be integrated into most SPH models straightforwardly, i.e., eq. 1.35 becomes

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij} + \sum_{j\in\mathcal{B}_\mathbf{x}} \frac{\psi_j}{\rho_j} A_j W_{ij}, \tag{1.39}$$

with $\mathcal{N}_\mathbf{x}$ and $\mathcal{B}_\mathbf{x}$ being neighboring fluid and boundary particles, respectively, and ψ_j is an artificial mass assigned to a boundary particle to correct for the sampling [Aki+12]. However, sampling boundary surfaces is a challenging problem and the sampling quality can strongly influence the fluid behavior [BGT17].

Regardless of the specific approach, particle-based approaches are a relatively natural extension of SPH dynamics and, consequently, the inclusion of two-way coupling can be done straightforwardly [Ihm+13]. Additionally, particle-sampled boundary geometries can also interact in a rigid-rigid fashion using the sampled particles [Gis+19], which makes these approaches attractive for Computer Animation. However, spatial adaptivity poses a significant problem to any explicitly-sampled boundary representation, as the representation needs to be locally adjusted based on the local fluid resolution to avoid erroneous behavior, e.g., particles penetrating the boundary; see the discussions in the paper reprinted in Chapter 6. While locally adjusting the resolution of the sampled boundary in a similar manner as for deformable rigid objects [Aki+13b] might be applicable, this approach has not been investigated exhaustively.

1.4.3 Wall-renormalization approaches

Wall-renormalization approaches include the boundary contribution indirectly as a corrective term for the fluid-only interpolant, i.e., a renormalization constant γ is determined by

$$\gamma(\mathbf{x}) = \int_{\Omega^f_\mathbf{x}} W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}', \tag{1.40}$$

for the local boundary domain $\Omega^b_\mathbf{x}$ where the fluid-only interpolation is inversely-scaled by this term, i.e., eq. 1.35 becomes

$$A(\mathbf{x}) \approx \frac{1}{\gamma(\mathbf{x})}\sum_{j\in\mathcal{N}_j}\frac{m_j}{\rho_j}A_jW_{ij}, \tag{1.41}$$

with the gradient terms

$$\nabla A(\mathbf{x}) \approx \frac{1}{\gamma(\mathbf{x})}\int_{\Omega^f_\mathbf{x}} A(\mathbf{x}')\nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}' + \frac{1}{\gamma(\mathbf{x})}\int_{\partial\Omega^B_\mathbf{x}} A(\mathbf{x}')\mathbf{n}W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}', \tag{1.42}$$

where $\partial\Omega^B_\mathbf{x}$ is the boundary surface and $\mathbf{n}$ is the surface normal. Evaluating these terms, however, is not trivial. Many different approaches, which commonly utilize some form of the divergence theorem [Lag61; Gau13; Ost28; Gre28], have been proposed in the past for this purpose, e.g., using semi-analytic approaches in 2D [Fer+13] and 3D [May+15], for incompressible methods [Ler+14] and using fully analytical approaches [Chi+19]. These methods can be computationally expensive, and the solution of the integrals is often limited to specific applications, dependent on the boundary element sizes and contain gradient terms that are not trivial to evaluate.

Within Computer Animation, these approaches have not been adopted widely but they offer an interesting avenue in future research, especially regarding spatial adaptivity. Chiron et al. [Chi+19] proposed an approach that is applicable regardless of the boundary element size, which could potentially be applied to spatially adaptive SPH simulations. However, integrating two-way coupling into these approaches is generally difficult and not as straightforward as with particle-based approaches.

1.4.4 Direct boundary integral methods

Direct boundary integral methods, in contrast to renormalization approaches, directly evaluate the integral form of the SPH interpolant across a boundary domain [FM15], making them fairly straightforward to integrate into existing SPH methods, i.e., eq. 1.35 remains unchanged:

$$A(\mathbf{x}) \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W_{ij} + \int_{\Omega^b_\mathbf{x}} \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}', \tag{1.43}$$

where the integral term is directly evaluated. Koschier and Bender [KB17] utilized a pre-determined grid of integral values, for computational efficiency, and local boundary contact points to implement two-way coupling, making the approach practical to use. The paper reprinted in Chapter 6 presents a semi-analytic boundary integral approach that does not require expensive computations beforehand and can be applied across varying resolution scales. However, the approach reprinted in Chapter 6 does not use an exact solution of the integral but uses an approximation based on an assumption of local flatness instead, which also increases the scale invariance of boundary geometries. Note that on a fundamental level, both integral and renormalization approaches solve very similar problems, despite being applied in a very different manner on the SPH interpolant.

1.4.5 Boundary pressure

In general, pressure forces in SPH are evaluated using a symmetric gradient formulation [Ihm+13], i.e., the following integral is evaluated over a local boundary domain $\Omega^b_\mathbf{x}$:

$$\nabla_\mathbf{x} p(\mathbf{x}) \approx \int_{\Omega^b_\mathbf{x}} \left[\frac{p(\mathbf{x})}{\rho(\mathbf{x})^2} + \frac{p(\mathbf{x}')}{\rho(\mathbf{x}')^2}\right] \nabla_\mathbf{x} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}'. \tag{1.44}$$

This term has to be evaluated for fluid particles, i.e.,

$$\nabla_i p_i \approx \int_{\Omega^b_i} \left[\frac{p_i}{\rho_i^2} + \frac{p(\mathbf{x}')}{\rho(\mathbf{x}')^2}\right] \nabla_\mathbf{x} W(\mathbf{x}_i-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}', \tag{1.45}$$

where several choices have been proposed to evaluate the boundary pressure term $p(\mathbf{x}')$. The most direct approach is to assume that $p(\mathbf{x}') = 0$, but this is generally not stable in incompressible approaches. The second most direct approach is pressure mirroring, i.e., $p(\mathbf{x}') = p_i$, where the pressure is assumed to be constant across the local boundary domain but different for each particle. This approach is straightforward to implement but causes inconsistencies in the pressure values.

The next most direct approach are interpolation methods which are generally only applicable to particle-based approaches. In these approaches, the pressure for each boundary element can be evaluated as if they were fluid particles [Ban+18a], and consequently, all boundary particles have consistent but different pressure values. While this is straightforward to implement since the same process is used for all particles, the resulting pressure field is not smooth at the boundary surface.

To solve this problem, Band et al. [Ban+18a] proposed a moving least-squares pressure extrapolation approach, where pressure values on boundary elements are not interpolated using SPH but extrapolated based on pressure values on fluid particles. This approach yields a significantly smoother pressure field, and can be applied to arbitrary positions, i.e., the extrapolation can be performed for non-particle-based boundary elements; see the paper reprinted in Chapter 6. Finally, several other approaches exist in the CFD community, but these have not been applied in a Computer Animation context and, consequently, will not be discussed.

1.5 Data handling and particle neighborhoods

The previous sections have only discussed mathematical and physical foundations of SPH, but implementing an actual SPH simulation also requires approaches to handle particle-based data and efficiently evaluate SPH interpolants. While many details of data handling are strongly platform specific, i.e., GPU and CPU-based SPH simulations have significantly different requirements for data structures, some notions still apply agnostically. In general, most SPH simulations utilize two data structures, where the first is a neighborlist per particle, which stores an explicit reference to all neighbors of a given particle, and the second is a lookup structure to find the region in memory corresponding to a region in space. Creating a neighborlist is relatively straightforward using an appropriate lookup structure, i.e., each particle needs to search for potential neighbors, and store a reference to each actual neighbor. While important for the computational performance of the simulation, these structures are highly platform dependent; this will be further elaborated on in the papers reprinted in Chapters 2, 4 and 5.

The main challenge in implementing a lookup structure is to consume as little memory as possible while allowing for fast access of memory based on the queried region in space. For example, a naïve lookup structure would refer any point in space to a single list containing all n particles, requiring $\mathcal{O}(n)$ accesses to memory for each query of a spatial region, which is computationally infeasible as the complexity of the entire SPH simulation would consequently become $\mathcal{O}(n^2)$. Accordingly, the goal is to provide a narrow memory range corresponding to each specific spatial location as this reduces the overall complexity to $\mathcal{O}(m \cdot n)$, where m is the average number of particles returned for each access.

For uniform resolution simulations, each particle only needs to access other particles within a global support radius h. This makes tiling the simulation domain into cells of size h an obvious approach, as this limits the maximum number of cells that are accessed per particle to 3^d. Using the arguments from the paper reprinted in Chapter 4, this also means that each cell contains on average $\frac{3}{4\pi}N_h$ particles, and each particle will access $\frac{81}{4\pi}N_h$ particles for each SPH interpolant in 3D. Storing these cells can be done in a variety of ways, e.g., using dense grids [Gre10], or compact hash maps [Ihm+14], which have added complexity but its memory requirements are only dependent on the number of particles, and are independent on the simulation domain size; see the paper reprinted in Chapter 4.

For simulations with variable support radii, the tiling should be performed using the maximum support radius of the simulation. However, this means that particles with smaller support radii would access more particles as potential neighbors. While this can be acceptable for non-spatially adaptive simulations, in spatially adaptive simulations with an adaptive ratio of α, each particle could access up to $\frac{81}{4\pi}\alpha N_h$ particles. This is computationally impractical for large adaptive ratios, e.g., above 100. Consequently, the paper reproduced in Chapter 4 proposed utilizing a self-similar space-filling curve to order the cells, a compact hashing approach, and constructing multiple lookup structures, for the same particle data, based on the different particle resolutions in the simulation. For more details on this approach, see the discussion in Chapter 4 and 5.

1.6 On stability, accuracy and error

Before discussing the main topic of this dissertation, namely spatial adaptivity, a discussion of common terminology that will be used in the evaluation of the reprinted papers, and in a general context of Computer Animation, is warranted. Within the CFD community the evaluation of results is mostly done using standardized benchmark cases, e.g., the lid-driven cavity test [Ler+14; Lee+08], dam breaking over a wedge [Ler+14; Fer+13], a moving square [Vac+13; Chi+19; Mar+13; Lee+08] and cylinder test [Vac+13; Taf+17; Chi+19; Mar+13], or against actual real world experiments [May+15; Fou+19; Chi+19], with a clear definition of terms coming from a numerical analysis background. Regardless, in a CFD context, the accuracy and stability of SPH have been ongoing topics of research for the past 20 years, see the recent survey from Vacondio et al. regarding grand challenges in SPH [Vac+21], with some recent work providing initial proofs for certain SPH variants on their solvability in special cases [Imo19].

Within Computer Animation, simulations are often described as being *vivid* [Liu+21], having *high fidelity* [Kug+21], as well as having less artifacts [Ban+18a; BK15], or use some vague notion of *stable* [Ihm+13; BK15; Liu+21; Ben+20], but these terms are generally not well-defined. While the discussion here cannot rectify this issue, it aims to make the evaluation of the reprinted papers more well-defined. Consequently, this discussion does not aim to define each of the above terms exactly, but instead proposes a consistent interpretation of these terms as they are used within this dissertation.

Throughout this dissertation *vivid*, or *high fidelity*, refers to a simulation approach that does not dampen high frequency details, e.g., the fluid surface contains significantly more detail; see Figure 7.12, which demonstrates this effect. Similarly, the *realism* of the simulation is not well-defined but tends to mean that a simulation looks like something that could happen in reality, which is strongly dependent on the reader's experience with real world fluid mechanics. Consequently, the terms *realism* and *realistic* are only used as abstract notions and not for quantitative evaluations.

Accuracy, in a general sense, refers to how close a calculated result is to some idealized outcome. Creating a ground truth reference solution for each simulation, using physical experiments, is prohibitively expensive and difficult in practice especially in Computer Animation which commonly involves large scale simulations. Therefore, the term accuracy instead describes how close a simulation is to results obtained solely based on underlying, fundamental, laws. These fundamental laws serve as guiding principles in the development and evaluation of new methods. The following principles, in no specific order, are generally relevant in a Computer Animation context:

- Incompressibility, i.e., $\rho \leq \rho_0$ [GEF15; BK15],
- Divergence-Freedom, i.e., $\frac{\partial \rho}{\partial t} = 0$ [BK15],
- Conservation of Mass, i.e., $\sum_{i=0}^{n-1} m_i$ is constant at all points in time [SG11; HS13],
- Conservation of Energy, i.e., $\sum_{i=0}^{n-1} u_i^{t+\Delta t} \leq \sum_{i=0}^{n-1} u_i^t$ [Ben+17; Ban+18a],
- Conservation of Momentum, i.e., $\sum_{i=0}^{n-1} m_i^{t+\Delta t} |\mathbf{v}_i^{t+\Delta t}| \leq \sum_{i=0}^{n-1} m_i^t |\mathbf{v}_i^t|$, without external influences [Ban+18a; Liu+21] ,
- Accuracy of the underlying discretization, i.e., $\langle A(\mathbf{x}) \rangle = A(\mathbf{x})$.

Note that momentum and energy are not perfectly conserved in most simulation approaches as they can be dissipated over time, e.g., kinetic energy can be transformed into thermal energy that is not explicitly tracked. An *error* is then defined as any deviation from these six guiding principles, e.g., a particle having a higher density than its rest density in an incompressible simulation. These errors can be spatially localized (i.e., the underlying cause is limited to individual particles) or systemic (i.e., the underlying cause affects groups of particles).

Stability then refers to the ability of a method to handle errors that occur during the simulation without causing the simulation to diverge significantly from the expected result that would have occurred without any errors during the simulation. For example, the boundary handling method reprinted in Chapter 6 causes a single particle to be stuck in concave corners of a simulation domain, but the overall simulation result is not negatively impacted by this. On the other hand, Fig. 7.15 demonstrates a simulation scenario where one evaluated method causes a visually apparent *explosion* to occur due to errors from refinement close to a boundary interface. Generally, an unstable simulation will be visibly unstable (e.g., through local explosions and unexpected behavior of the fluid) and/or have significant numerical anomalies (e.g., a particle suddenly accelerates from an expected velocity to a velocity much higher than any other particle). A *stable* simulation is therefore any simulation that remains accurate, within some deviation from the desired result, during the simulation. Consequently, a simulation method's *stability* describes how many situations the method remains stable in. A method is considered as *more stable* than another method if it remains stable in a larger set of situations, e.g., through a reduction of the magnitude of common errors or through an increased resilience towards errors.

Understanding what kinds of errors (localized or systemic) can occur from underlying causes related to the six guiding principles is crucial. Consequently, a brief discussion on how each guiding principle can be violated through errors in spatially adaptive and incompressible simulations is warranted. Note that these discussions are based on practical experience as well as empirical evaluations, and are therefore not an exact analysis of all possible sources and errors.

Discretization accuracy is mainly limited by this transformation of eq. 1.2 into eq. 1.3, i.e., the transformation

$$\int_\Omega \frac{A(\mathbf{x}')}{\rho(\mathbf{x}')} W(\mathbf{x}-\mathbf{x}',h)\rho(\mathbf{x}')d\mathbf{x}' \approx \sum_{j\in\mathcal{N}_\mathbf{x}} \frac{m_j}{\rho_j} A_j W(\mathbf{x}-\mathbf{x}_j,h). \tag{1.46}$$

As mentioned before in Sec. 1.1, this discretization is primarily dependent on the particle distribution. In general, an isotropic particle distribution yields the numerically best discretization for a spherical support domain. Particle-shifting techs aim to improve this isotropy, and have found wide usage in weakly compressible simulations in the past [Sun+19; VR17]. However, only recently have particle-shifting techniques that do not violate momentum conservation for incompressible simulations been proposed [KGS19]. Lind et al. [LRS20] provided a broad survey on the discretization accuracy, especially with regards to particle shifting, in SPH from a CFD context.

However, the particle distribution for spatially adaptive simulations is inherently non-isotropic, i.e., the transition from regions with lower resolutions to those with higher ones necessarily implies the existence of non-isotropic distributions. These errors can therefore occur everywhere in the simulation, especially on resolution interfaces, potentially causing significant problems throughout the simulation. While no particle-shifting method or modified SPH model has been proposed to address this problem so far, practical experience has shown that avoiding large resolution gradients and allowing for smooth transitions of resolution can greatly improve spatial adaptivity. For more details on this, see the paper reprinted in Chapter 3.

Conservation of mass for a uniform simulation is, in general, trivially possible as the overall mass of the simulation depends solely on the combined mass of all particles, which generally remains unchanged. For adaptive simulations, that do not use adaptive approaches that explicitly remove particles [SG11], the only error related to mass conservation that can occur are numerical mass rounding errors due to repeated splitting and merging of particles. However, this error is generally small and can be ignored as in most situations as the accumulation only becomes significant over periods longer than typical simulations, e.g., after millions of changes of resolution.

The ***conservation of energy and momentum*** is a two-fold problem:

From a model perspective, if only momentum and energy-conserving methods are utilized, e.g., a momentum-conserving viscosity model is used [Pri12], then the total momentum of the simulation should not increase. Furthermore, if all external effects, e.g., gravity, are accounted for, then the total energy in an accurate simulation should not increase if energy-conserving models are used throughout.

Errors due to any violation of the other guiding principles are not of a physical nature and have to be considered separately. Notably, these errors are a symptom of an underlying error, e.g., a violation of incompressibility, and not the root cause. For example, if any part of the simulation yields a density error, e.g., two particles are closer than they should have been under the given simulation parameters, then an excessive pressure may be generated. However, as this pressure should not exist in a perfectly accurate simulation, the pressure can generate forces that introduce energy and momentum into the simulation. If enough energy is added to the simulation then the overall simulation can be destabilized. Consequently, a violation of conservation of energy and/or momentum, is generally not the root cause of an error but an indicator than an error has occurred, which resulted in this violation. Accordingly, monitoring the total energy and momentum of a simulation can be an effective way to detect potentially destabilizing errors.

Density errors are generally a symptom of problems that occurred prior, i.e., if the density is above the rest density then this is generally, barring errors due to violations of the discretization accuracy, the result of pressure forces that were calculated in a prior timestep. Density errors mainly arise from two distinct sources: (i) the convergence criterion of the solver, and (ii) the density prediction step.

The former problem is generally the result of the convergence criterion not being set to result in a *zero* error, but instead yielding a small but acceptable error. This is reflected in practice, where a threshold of 0.01% for the density error and 0.1% for the divergence error are considered to be sufficient for a simulation to be incompressible, in the Computer Animation community [BK15]. Consequently, the error introduced here is systemic and choosing a significantly less stringent convergence criterion, e.g., a 10% density error, may yield an unstable simulation. This kind of error can be limited by using the aforementioned convergence criteria.

Errors in the density prediction step are significantly more challenging to handle. This error arises from the usage of a semi-implicit Euler scheme, i.e., only first-order derivative terms are used for predictions, and from the usage of a compact support radius h. Both choices inherently yield an error in the predicted density of the new timestep, i.e.,

(1.47) $$\rho^{t+\Delta t}(\mathbf{x}) \neq \rho^t(\mathbf{x}) + \Delta t \frac{d\rho^t(\mathbf{x})}{dt},$$

which can be significant. The error from this prediction is dependent on the relative position of two particles $|\mathbf{x}_i - \mathbf{x}_j|$ and their relative velocity $\frac{d|\mathbf{x}_i - \mathbf{x}_j|}{dt}$, as shown in Fig. 1.1. This error is mostly determined based on the relative motion between two particles combined with the timestep Δt. While a single particle can move at most $0.4h$ per timestep using a standard CFL condition [Ihm+13], two particles moving directly towards each other would reduce the distance between them by up to $0.8h$ per timestep. As a result, the error in the density estimate can be significant and errors of around 80% are not uncommon in practice, even in uniform resolution simulations. Predictive-corrective incompressible SPH [SP09] avoids this issue by evaluating a predicted density value using an explicitly integrated simulation, i.e., the method evaluates $\rho^{t+\Delta t}(\mathbf{x}^{t+\Delta t}) \approx \rho^t(\mathbf{x}^t + \Delta t \mathbf{v}^t)$ instead of $\rho^{t+\Delta t}(\mathbf{x}^{t+\Delta t}) \approx \rho^t(\mathbf{x}^t) + \Delta t \frac{d\rho^t(\mathbf{x}^t)}{dt}$. However, in practice, errors still occur in this approach as the neighborlists are not necessarily updated at each prediction.

Overall, these prediction errors significantly violate the assumption of incompressibility and cause localized density errors that need to be corrected in the next solver iteration, which leads to excessive pressure values counteracting this prediction error. On a fundamental level, such a reaction will add additional energy to the system, which needs to be compensated for to ensure the conservation of the total energy. The usual approach in compensating for this additional energy is by introducing an artificial viscosity term to stabilize the simulation. This is most commonly done by smoothing the velocity field [Mon02], as this dissipates the introduced energy spatially throughout the simulation. If the artificial viscosity is

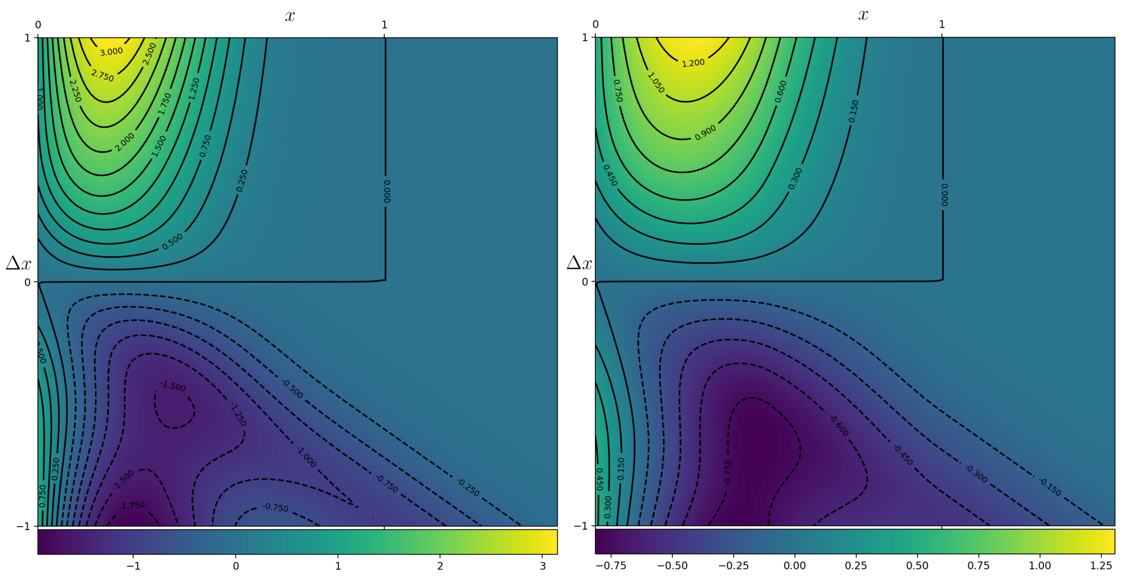


Figure 1.1: This figure shows the error in the estimate of the kernel function using a first-order Taylor series. Accordingly, the figure shows $f(x, \Delta x) = \hat{W}(|x - \Delta x|) - \hat{W}(x) + \Delta x \frac{\partial}{\partial x}\hat{W}$, with $\hat{W}$ defined in eq. 1.14 and $x, \Delta x \in \mathbb{R}$. The left subfigure shows the error for the Wendland-4 kernel and the right subfigure shows the error for the cubic spline kernel.

set to be too low, however, then the dissipation of energy will be insufficient in preventing larger stability issues. Furthermore, if the correction of the density error itself introduces significant errors then the overall errors can quickly diverge in magnitude and yield an unstable simulation.

Within the context of an incompressible fluid simulation, which will be used throughout this dissertation, two general kinds of prediction errors occur, each having its own causes: errors that are uncorrelated in their magnitude and occur in single particles or clusters of particles, and errors that are correlated in their magnitude or share an underlying cause, e.g., the overall fluid's behavior. Accordingly, these errors can be classified based on whether the relative motion between particles is relatively similar, or are significantly different across particles with an underlying shared cause.

On one hand, assuming that the velocity field of the simulation is fairly smooth, the differences in velocity between particles will result in different errors in the density estimate for different particles. Accordingly, this error can be modeled as pseudo-random noise on the density estimation error. This error can then readily be compensated for through artificial viscosity, which smoothens out the resulting noisy velocity field.

On the other hand, larger scale events in a fluid simulation, e.g., a wave collapsing into a stationary fluid bulk, yield significant differences in the relative motion between particles, which are correlated through an underlying common event. These problems are further exacerbated in vortices where the local rotation of the fluid causes significant relative motion of particles in a region. Such errors are difficult to compensate for, and the most straightforward solution is limiting the overall timestep, which in turn limits the maximum error in the density estimate.

To summarize, density errors significantly contribute to the overall simulation error, either directly or indirectly, and any spatially adaptive simulation should avoid introducing significant errors in the density field. Furthermore, simulation settings should be chosen reasonably, e.g., if a collision of two streams flowing in opposite directions is to be simulated, then it is generally better to chose a smaller timestep than the one indicated by the CFL condition before the collision.

1.7 Spatial adaptivity

As mentioned previously in the introduction, global changes in resolution are challenging due to the required computational resources. Accordingly, spatial adaptivity is an attractive alternative approach that limits the increases in resolution only to areas where this increase is most beneficial, e.g., only at the fluid surface. However, directly adjusting the position or size of individual particles could introduce significant sources of errors, e.g., combining two particles to reduce the local spatial resolution can readily yield a change in the density field of the fluid. Thus, a key point in this dissertation is finding approaches to enable spatial adaptivity whilst obeying the principles discussed in Section 1.6. In this regard, this section provides an overview of the main concepts pertaining spatial adaptivity, its applications within SPH and how errors occurring during changes in resolution can be diminished.

Spatial adaptivity is at the center of the research discussed within this dissertation and the underlying methods used in this field make heavy use of all of the aspects of SPH discussed in prior sections. In general, spatial adaptivity describes methods that, during a simulation, change the spatial resolution of the simulation itself. These changes can yield significant sources of errors, as they were discussed prior, as directly modifying the position or size of individual particles can introduce arbitrarily large errors. In this regard, this section will give an overview of the main concepts of spatial adaptivity, as it is applied within SPH, and how errors occurring during the change of resolution can be diminished.

The most common application of spatial adaptivity is where the simulation resolution is either refined to a higher resolution or merged to a lower resolution. Consequently, *adaptivity* is, as per the paper reprinted in Chapter 7, a multi-step process that involves:

- a sizing function to determine the desired resolution of a particle,
- merging a fine particle with other particles,
- smoothing the particle resolution gradient,
- refining a coarse particle into multiple finer particles, and
- methods to limit the impact of errors caused by the steps above.

Changing the spatial resolution in a particle-based simulation requires adjusting individual particles, i.e., combining, splitting and redistributing them. Based on Sec. 1.6, a set of guiding principles should be used as a guideline for any method. Consequently, a spatially adaptive method should (i) not introduce discretization errors, (ii) obey conservation of mass, (iii) obey conservation of linear and angular momenta as well as energy, and (iv) not introduce density errors.

The introduction of discretization errors is mainly dependent on the sizing function. A good sizing function should yield smooth transitions between areas of different resolutions, and the mechanisms to adjust the resolution should yield an actual resolution close to the sizing function. The mechanisms for changing the resolution can also introduce other problems, where the density error is generally the most significant problem. Consequently, the density at a point directly before and after a change in resolution can be defined as ρ and $\rho^\star$, respectively, yielding an error metric at all points in space $\mathbf{x} \in \mathbb{R}^d$ given by

$$\epsilon(\mathbf{x}) = \rho^\star(\mathbf{x}) - \rho(\mathbf{x}), \tag{1.48}$$

see the discussions by Barcarolo et al. [BOD14] and Vacondio et al. [Vac+13], as well as the paper reprinted in Chapter 7. Moreover, every particle affected, or inserted, by a change in resolution has $2d+2$ degrees of freedom, i.e., its position, velocity, mass and support radius, which need to be adjusted in accordance with the guiding principles in Sec. 1.6. Note that while in CFD contexts the support radius is commonly optimized, in Computer Animation contexts the support radius is generally set based on the particle mass, which means that there are only $2d+1$ degrees of freedom per particle.

1.7.1 Particle sizing

The goal of a particle sizing function $S(\mathbf{x})$ is to determine what resolution a point in space $\mathbf{x}$ should be simulated with, i.e., what size the particle should have, based on the importance of its location for the desired outcome of the simulation. Furthermore, based on the guiding principles discussed in Sec. 1.6, $S(\mathbf{x})$ should yield a smooth transition from areas that are very important (i.e., areas where a higher resolution is used) to areas that are less important (i.e., areas where a lower resolution is used). Accordingly, the first step is finding a metric to decide if some part of the fluid is important. In Computer Animation, particles at the fluid surface have the highest importance as the surface has the largest impact on the visual appearance of a fluid. Detecting the fluid surface in SPH however, especially for adaptive simulations, is not a trivial problem and various approaches for detecting the fluid surface have been proposed, e.g., [Mar+10; Ort+13; BTN13]. To create a smooth resolution gradient, a distance metric describing how far each particle is from the closest area of high interest can be defined, e.g., the particle's distance to the visible fluid surface, which is then used as the basis of $S(\mathbf{x})$. In the papers reprinted in Chapters 3 and 7, the approach by Barecasco et al. [BTN13] is used to detect the fluid surface, using a simple geometric construction process. This is combined with the iterative distance approach of Horvath and Solenthaler [HS13] to determine the distance of each fluid particle from the visible fluid surface.

Given a distance $d(\mathbf{x})$ to an area of interest, an adaptive ratio $\alpha = \frac{V_{\text{base}}}{V_{\text{min}}}$ of the volume of smallest particle, V_{min}, to that of the largest particle, V_{base}, and a maximum user-defined distance d_{max}, a sizing function can be constructed as

$$V(\mathbf{x}) = \left[\frac{1}{\alpha} + \frac{d(\mathbf{x})}{d_{\text{max}}}\left(1 - \frac{1}{\alpha}\right)\right] V_{\text{base}}. \tag{1.49}$$

This sizing function $V(\mathbf{x})$ determines the desired particle volume for a given spatial location as a linear interpolation between the finest resolution $\frac{V_{\text{base}}}{\alpha}$ at the fluid's surface, and the coarsest resolution V_{base} at the user-defined maximum distance d_{max}; see Sec. 9 of the paper reprinted in Chapter 7.

Consequently, considering the errors induced on the simulation, this sizing function should be temporally coherent, i.e., desired resolutions should change smoothly over time, and changes in resolution should not suddenly occur over large regions at once, to avoid introducing correlated clusters of errors. However, there are two straightforward cases that violate these restrictions.

Firstly, if a wave, or other structure, collapses into the fluid, the fluid surface of that region disappears, and consequently the region becomes less important as a whole and a large amount of fluid needs to change its resolution simultaneously. This problem can be mitigated through a temporal smoothing of the distance metric, as employed by Horvath and Solenthaler [HS13], as this staggers the change of resolution over time. However, these situations are also a general source of instability in spatially adaptive SPH methods. Hence their impact should also be reduced either by temporally smoothing the change of resolution driven by the sizing function, e.g., through temporal blending [Ort+13], or using other ways to produce more gradual changes in resolution; see also the papers reprinted in Chapters 3 and 7.

Secondly, generating initial particle configurations is challenging, even in uniform resolution simulations; see [Die+15]. Accordingly, in Computer Animation, the most common approach is to start with an initially uniform fluid resolution and only enable spatial adaptivity at some later point during the simulation. However, this could lead to the entire fluid volume changing its resolution at once, which in turn would yield significant correlated errors that need to be accounted for. Therefore, enabling spatial adaptivity should be performed in gradual steps, e.g., α is increased over time. Additionally, to avoid the errors induced by resolution changes from compounding with simulation stability concerns, spatial adaptivity should be enabled before any challenging simulation scenarios are expected to occur.

1.7.2 Decreasing particle resolution

Merging denotes the process in which a set of given particles $j \in \mathcal{M}$ is combined to form a single particle i. To enforce the constraints based on the guiding principles from Sec. 1.6, the mass of the resulting particle should be equal to the mass of the combined particles [BOD14], i.e.,

$$m_i = \sum_{j \in \mathcal{M}} m_j. \tag{1.50}$$

Then a mass weighted average position and velocity, which enforces conservation of momentum, is determined as

$$\mathbf{x}_i = \frac{\sum_{j \in \mathcal{M}} m_j \mathbf{x}_j}{m_i}, \ \mathbf{v}_i = \frac{\sum_{j \in \mathcal{M}} m_j \mathbf{v}_j}{m_i}. \tag{1.51}$$

Each merging process also removes the influence of the merged particles $\mathcal{M}$ and adds a single new particle, i.e., there exists a difference term, related to the density field, given by

$$\epsilon(\mathbf{x}) = m_i W(\mathbf{x} - \mathbf{x}_i, h) - \sum_{j \in \mathcal{M}} m_j W(\mathbf{x} - \mathbf{x}_j, h). \tag{1.52}$$

Note that the above equations describe m_i, $\mathbf{x}_i$ and $\mathbf{v}_i$ exactly, and therefore the only remaining free parameter is the support radius of the new particle h_i. Consequently, the only way to minimize the difference term in eq. 1.52 is by modifying the support radius of the merged particle, see Barcarolo et al. [BOD14]. However, within Computer Animation the support radius is defined solely using the particle mass and, consequently, this optimization cannot be applied and the error due to merging cannot be minimized.

In practice, determining the degrees of freedom of a merged particle is not the main issue. Rather, the biggest challenge arises from the difficulty in finding particles that are eligible to be merged, as this requires either pairs, or groups, of particles that are spatially close to each other with a combined mass below the splitting threshold. Furthermore, not all pairings of particles yield good results when merged, i.e., the resulting merged particle could be placed at the location of another particle, which yields significant errors in the density field. Accordingly, several strategies have been proposed to find appropriate particle pairings; see the discussions in Chapter 2 for a more in-depth discussion.

In this context, $n : n-1$ merging is also relevant, where a single small particle is merged evenly into a set of other particles. This can conceptually be treated as a merging of the small particle, which was split into $n-1$ parts, with each of the other particles separately. More complicated merging processes, e.g., $n : m$, are an interesting avenue for further research to reduce the overall errors introduced and smoothen the resulting resolution gradient without affecting the support radii. Note that as long as suitable particles are chosen, i.e., the resulting merged particle is not spatially close to another existing particle, the error induced in this process is negligible in practice. However, if no merging is performed at all, i.e., differently sized particles can mix freely, the resulting particle distributions tend to be numerically problematic and computationally expensive, see Sec. 1.6 regarding the errors in the discretization.

1.7.3 Smoothing particle resolution

Sharing is a process in which a set of particles $j \in \mathcal{S}$ redistribute mass and other quantities to smoothen out the resolution of the fluid. This is mostly applicable to particles that only slightly deviate from their ideal size, based on a sizing function, where neither merging nor splitting of the particle would be appropriate. Accordingly, a particle that is too large, relative to its desired size, can be pseudo-split into two particles at the same location with one particle at the desired size and one particle to be merged into other particles. Consequently, this process is analogous to merging and shares the same problem of finding appropriate pairs of particles. This is discussed in more detail in the paper reprinted in Chapter 3.

1.7.4 Increasing particle resolution

Splitting is a process in which a single coarse particle i is replaced with a set of refined particles $j \in \mathcal{R}$. The problem has $d+2$ degrees of freedom per refined particle, i.e., the positions, masses and support radii of the refined particles can be adjusted freely, whereas the velocity is fixed to ensure conservation of momentum [Fel06]. This problem contains a significant number of degrees of freedom which makes placing the refined particles into the simulation difficult. In this regard, there are two main problems: (i) the particles need to be inserted in some spatial configuration and (ii) the mass and support radii need to be optimized for each inserted particle. Overall, the refinement process should minimize the same point-wise error as the merging process, i.e.,

$$\epsilon(\mathbf{x}) = m_i W(\mathbf{x} - \mathbf{x}_i, h) - \sum_{j \in \mathcal{R}} m_j W(\mathbf{x} - \mathbf{x}_j, h), \tag{1.53}$$

is to be minimized over the entire simulation domain. This yields a minimization problem, under a constraint driven by conservation of mass, as

$$\min_{\mathbf{x}_{\mathcal{R}}, m_{\mathcal{R}}, h_{\mathcal{R}}} \int_{\mathbb{R}^d} \epsilon(\mathbf{x})^2 d\mathbf{x}, \; \sum_{j \in \mathcal{R}} m_j = m_i, \tag{1.54}$$

Inserting new particles at mostly random positions can work for weakly compressible SPH methods [Ada+07], but the introduced error is uncontrolled and can yield local errors that are large enough to destabilize an incompressible fluid simulation. Accordingly, fixed patterns of particle configurations, i.e., so-called refinement patterns, have found wide usage in the past as they can be optimized for a low induced error. For example, Orthmann and Kolb used a 1:2 refinement pattern [OK12], Solenthaler and Gross used a 1:7 refinement pattern [SG11] and Vacondio et al. proposed using a 1:13 pattern [Vac+13]. In contrast to this, the paper reprinted in Chapter 3 utilized refinement patterns for 2 to 16 particles. However, these patterns are commonly applied equally across all particles, which can lead to stability issues in the simulation as any error inherent to the refinement process would not be locally confined. Initial work in avoiding this behavior was done by Adams et al. [Ada+07], who utilized randomized refinement processes, as well as Horvath and Solenthaler [HS13], who randomly rotated the refined particles.. The paper reprinted in Chapter 7 performs an online statistical optimization of the refined particles during the actual simulation to reduce the local error and prevent introducing the same error everywhere. This method also removes the need to intuitively find an initial refinement pattern as the optimization process can be performed a priori using randomized inputs and the online optimization can be started with these pre-optimized patterns, instead of manually tuned patterns.

With the positions of particles determined using either refinement patterns or local optimization, two degrees of freedom remain, i.e., mass and support radii. Considering mass, most approaches utilize a fixed uniform distribution of mass among the inserted particles [OK12; HS13; WHK17]. However, optimizing this distribution of masses can yield a reduced error [FB07]. Optimizing the distribution of masses is usually done a priori, i.e., when the refinement patterns are created, and under the assumption of an isotropic particle distribution [Vac+13]. In addition to this optimization, the paper reprinted in Chapter 7 performs an online optimization of the mass distribution during the actual simulation to reduce the error inserted into the simulation. In the same way, support radii can be optimized to further reduce the error induced through particle refinement, but this is not done in Computer Animation as support radii are directly coupled to mass.

The error induced by the refinement patterns can be controlled reasonably well by using patterns that are not adjusted to the local fluid domain. However, this is not the case when boundary objects are present. If a particle that should be refined is in contact with a rigid object then placing the refined particles as if the boundary object does not exist can yield significant errors, see the evaluation in the paper reprinted in Chapter 7. This can be avoided indirectly by ensuring that particles interacting with boundary geometries have the appropriate resolution, as done for example in the paper reprinted in Chapter 3. However, this is not an ideal solution as it requires knowing where fluid-boundary interactions can occur and when they occur before they occur. Instead, by using the online optimization process proposed in Chapter 7, the refined particles can account for the boundary geometry and, consequently, reduce the induced error from refinement.

1.7.5 Error dampening measures

Particle refinement is generally the largest source of errors, and while it is possible to optimize the error due to refinement, the resulting error is never zero, i.e., some residual error will always remain, see the evaluation section of the paper reprinted in Chapter 7. However, this residual error can be minimized through optimized refinement patterns and online optimization. Consequently, the better the refinement process used, the smaller the error compensation required. This compensation can be achieved by introducing additional viscosity, or other smoothing effects, but it still needs to be compensated for to avoid spurious instabilities in the simulation. To account for the induced error, three distinct approaches have been applied in Computer Animation: (i) increase the artificial viscosity, (ii) remove particles with significant errors from the simulation, and (iii) temporally blend newly refined particles into the simulation.

Increasing the global artificial viscosity of a simulation aims to reduce the impact of spontaneous errors induced through refinement. While this does compensate for many errors, see the paper reprinted in Chapter 3, the artificial viscosity also dampens the fluid behavior in the entire simulation, even in the absence of changes to the spatial resolution. Accordingly, the paper reprinted in Chapter 7 proposes a local viscosity term that depends on the time since the most recent refinement for viscosity interactions involving refined particles. This can be achieved straightforwardly by scaling the viscosity parameter per particle interaction to be higher than normal, and then reducing this scaling as the simulation progresses. Consequently, spontaneous errors due to refinement experience additional dampening without requiring a significant global increase in dampening.

The second approach is to simply remove particles that are causing significant errors. This approach was first proposed by Solenthaler and Gross [SG11] and later refined by Horvath and Solenthaler [HS13]. The general idea in these methods is that simulations of different scales are coupled to each other and resolution changes involve moving the boundaries between the domains. In this approach, redundant particles get emitted during refinement, and particles that cause the largest density errors are removed until the overall desired number of created particles is reached. Note that this requires relatively fixed boundaries between resolutions, i.e., different resolutions cannot freely mix, and additional relaxation steps are still required to stabilize the resolution transition.

The final approach is to blend in new particles smoothly over time, i.e., the resolution is not changed instantaneously but over a short period of time. This approach was first proposed by Orthmann and Kolb [OK12], where the original particle that was refined is still explicitly tracked throughout the simulation as it interacts with other particles as if it was still a real particle. However, this explicit blending approach first requires all SPH interpolations to be performed on the original and refined particles and then, in an additional step, blending these particles together using a blend weight that decays over time, i.e., the influence of the original particle eventually goes to zero. The trajectory of the refined particles and the original particle are also averaged with each other to help in the blending process. While this process produces good results, tracking the original particle and performing the blending explicitly after each SPH interpolation is computationally inefficient. Consequently, the paper reproduced in Chapter 3, proposes an implicit blending approach that only utilizes the temporal blending approach for the density of the refined particles and that does not explicitly track the original particle. This method is then further improved through an optimization of the blend weights in the paper reproduced in Chapter 7.

1.8 Scope of work

Formally, this work thus presents six recently published, internationally peer-reviewed publications [WHK16; WHK17; WK19; WAK20; WK20; WK21], in each chapter based on their original text, using the respective author or open-access versions, with updated and unified formatting and minor corrections of linguistic problems. Furthermore, references to supplementary materials are either replaced with a reference to the definite version of the paper or resolved through inclusion of the supplementary material as an appendix. Every chapter is preceded by a brief contextualization that places the paper in context of the overall scope of research presented in this dissertation and delineating the contributions of all involved authors. In this manner the dissertation follows the statutory provisions for a cummulative dissertation laid out in the Promotionsordnung der Naturwissenschaftlich-Technischen Fakultät of the University of Siegen.

List of publications

[1] Rene Winchenbach, Hendrik Hochstetter and Andreas Kolb. Constrained Neighbor Lists for SPH-based Fluid Simulations. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), pp. 49-56, July 2016, ISBN: 9783905674613

[2] Rene Winchenbach, Hendrik Hochstetter and Andreas Kolb. Infinite Continuous Adaptivity for Incompressible SPH. In ACM Transactions on Graphics (TOG), volume 36 number 4, pp. 102:1-102:10, July 2017, doi: 10.1145/3072959.3073713

[3] Rene Winchenbach and Andreas Kolb. Multi-Level-Memory Structures for Adaptive SPH Simulations. In Proceedings of Vision, Modeling and Visualization (VMV), October 2019, doi: 10.2312/vmv.20191323

[4] Rene Winchenbach and Andreas Kolb. Multi-Level Memory Structures for Simulating and Rendering Smoothed Particle Hydrodynamics. In Computer Graphics Forum (CGF), volume 39 number 6, pp. 527-541, September 2020, doi: 10.1111/cgf.14090

[5] Rene Winchenbach, Rustam Akhunov and Andreas Kolb. Semi-Analytic Boundary Handling Below Particle Resolution for Smoothed Particle Hydrodynamics. In ACM Transactions on Graphics (TOG), volume 39, number 6, pp. 173.1:173:17, December 2020, doi: 10.1145/3414685.3417829

[6] Rene Winchenbach and Andreas Kolb. Optimized Refinement for Spatially Adaptive SPH. In ACM Transactions on Graphics (TOG), volume 40, number 1, pp. 8.1-8.15, January 2021, doi: 10.1145/3363555

Chapter 2

Constrained Neighbor Lists for SPH-based Fluid Simulations

Contextualization

This chapter reprints the publication “Constrained Neighbor Lists for SPH-based Fluid Simulations” published as a conference proceeding of the Symposium on Computer Animation (SCA 2016) [WHK16] with co-authors Hendrik Hochstetter and Andreas Kolb. This work represents the first steps towards the later adaptive SPH simulations through its application to multi-scale fluid simulations as a method that performs local optimizations of fluid quantities, in this case motivated by memory constraints, to improve the simulation performance, which coincidentally also helped to improve simulations stability in adaptive scenarios.

Conceptually, the focus of this paper is on constraining the maximum number of neighbors a particle can have. There exists a theoretical ideal number of neighbors per particle that is reached if a particle is in an isotropic particle neighborhood and at rest density; however, these conditions rarely occur and, especially with adaptive simulations, the number of actual neighbors of a particle can differ significantly. This turns out to be a significant problem for GPU-based simulations as storing reference to all neighbors of all particles, i.e., so called neighborlists, either involves over-allocating storage for all particles, to ensure that all particles can store their maximum number of neighbors, or using non coalescing data storage formats. These restrictions either limit computational performance, in case of non-coalescence, or limit the maximum number of neighbors excessively through memory requirements. This problem is addressed in this paper by reducing the support radius of particles with too many neighbors, relative to the allocated memory, until their number of neighbors is small enough.

The initial motivation of this paper arose from the collaboration with Hendrik Hochstetter on GPU-based SPH simulations as existing methods to handle neighborlists did not work well on GPUs and overly limited either performance or simulation scale. The main idea of enforcing maximum neighborlist lengths via adjusted support radii came from Rene Winchenbach with help and support in validating the approach by Hendrik Hochstetter. Hendrik Hochstetter supported the development of the concept and gave hints related to its implementation. Moreover, together with Andreas Kolb, he co-authored, i.e., he contributed various suggestions and assistance in structuring and writing of the final paper.

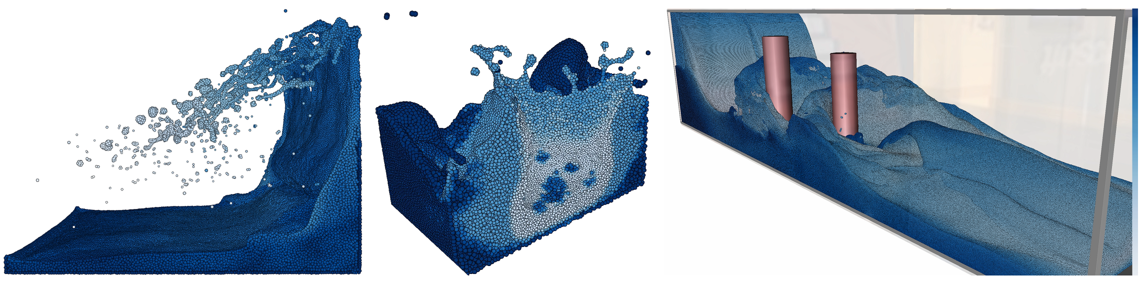


Figure 2.1: Our improved neighbor algorithm can handle real time simulations of over 500K particles (left image, velocity color coded), multiple fluid resolutions at once (middle image, support radius color coded), and large scale simulations with over 35 million particles (right image, velocity color coded).

Abstract

In this paper we present a new approach to create neighbor lists with strict memory bounds for incompressible Smoothed Particle Hydrodynamics (SPH) simulations. Our proposed approach is based on a novel efficient predictive-corrective algorithm that locally adjusts particle support radii in order to yield neighborhoods of a user-defined maximum size. Due to the improved estimation of the initial support radius, our algorithm is able to efficiently calculate neighborhoods in a single iteration in almost any situation. We compare our neighbor list algorithm to previous approaches and show that our proposed approach can handle larger particle numbers on a single GPU due to its strict guarantees and is able to simulate more particles in real time due to its benefits in regard to performance. Additionally we demonstrate the versatility and stability of our approach in several different scenarios, for example multi-scale simulations and with different kernel functions.

2.1 Introduction

The *Smoothed Particle Hydrodynamics (SPH)* method plays an important role in scientific computing and computer animation. Due to it's nature as a Lagrangian simulation it offers high spatial flexibility and support the simulation of incompressible fluids with free surfaces and various physical properties. In SPH fluids are described by unstructured particle data and local fluid quantities are interpolated from a set of particles within a compact support radius.

As these particle pairings need to be checked for every interaction it can be beneficial to store them in a neighbor list. Creating these neighbor lists has traditionally been very expensive on GPUs due to unbounded memory consumption and irregular access patterns; see [Ihm+11]. But they can still be used on GPUs if they are reused often in an iterative pressure solver;see [GEF15].

In this paper we introduce an efficient and versatile neighbor list method for incompressible SPH fluids simulations on GPUs with strict memory bounds and improved access patterns providing benefits in performance in all situations. To achieve this we propose a new formulation to locally adjust the particle support radius in every time step instead of using a fixed support radius. In order to guarantee our strict memory bounds we propose a predictive-corrective algorithm that correctly reduces the support radius of particles that violate the given bounds until they are correctly limited. Finally we propose a new structure to store the neighbor list in that improves access patterns and speeds up the overall simulation.

Using our proposed algorithms we are able to simulate larger particle sets on a GPU and perform faster calculations when comparing it to previous approaches. Additionally we show how our method is able to handle multiple fluid resolutions and dynamic rigid boundaries.

2.2 Foundations and related work

Since the introduction by Gingold, Monaghan [GM77] and Lucy [Luc77] in the field of astrophysics, SPH has spread into many areas of research including, our area of interest, computer graphics [MCG03]. First designed for the simulation of compressible fluids, SPH has since been extended to support incompressible fluids [MM13; Ihm+13; BK15], strong surface-tension effects [AAT13], two-way-interactions with rigid bodies [Aki+12] and many more effects. We refer the reader to the survey paper by Ihmsen et al. for a general overview [Ihm+14].

In SPH, fluid quantities are evaluated by interpolating information of neighboring particles. The interpolant of a quantity A of particle i at its position $\boldsymbol{x}_i$ depends on the position $\boldsymbol{x}_j$, mass m_j, density ρ_j and quantity A_j of the neighboring particles j and is commonly written as

$$A(\boldsymbol{x}_i) = \sum_j A_j \frac{m_j}{\rho_j} W(x_{ij}, H), \tag{2.1}$$

where $x_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ represents the distance between particles i and j [Mon05]. W is a kernel function that weights particle quantities based on x_{ij} and the support radius H. Interactions only take place if x_{ij} is shorter than the support radius H leading to a compact support radius. In practice, the dynamic particle volume $\frac{m_i}{\rho_i} = \widetilde{V}_i$ can be replaced by $\widetilde{V}_i = \frac{1}{\delta_i}$, where $\delta_i = \sum_j W(x_{ij}, H)$ is the particle density, in order to correctly handle density contrasts [SP08]. To stably handle free surfaces, Orthmann et al. presented another derivation based on the particle number density $n_i = \sum_j V_j W(x_{ij}, H)$ as $\widetilde{V}_i = \frac{V_i}{n_i}$, where $V_i = \frac{m_i}{\rho_{i,0}}$ is the particle's rest volume and $\rho_{i,0}$ its rest density [Ort+13].

The support radius either is uniform for all particles [Ihm+11; AAT13; Aki+12] or gets locally and dynamically adjusted for each particle to increase the simulation stability [Mon05; DA12] or in order to reduce the simulation time by only simulating at full particle resolution in areas of interest [SG11; OK12]. A common approach to calculate local support radii is given as

$$H_i = s_h \eta \left(\frac{m_i}{\rho_i}\right)^{\frac{1}{3}}, \tag{2.2}$$

where η is a configuration parameter set ideally somewhere between 1.2 and 1.3 [Mon05]. Note, we directly adjust the support radius in Eq. 2.2 instead of the smoothing length h_i [Mon05]. To be more consistent, we will only use the support radius throughout the text which is related to the smoothing length by the constant smoothing scale $s_h = \frac{H}{h}$ and depends on the shape of the kernel function [DA12]. In order to conserve quantities and momentum, particle interactions have to be symmetric. Thus, varying support radii are usually symmetrized as $H = \frac{H_i+H_j}{2}$ [Mon92; OK12]. Locally and dynamically adapting the support radius also introduces additional gradient H terms in each derivative. These additional terms, however, have neglectable effect [HK89], thus, they are usually omitted [HK89; OK12]. Although locally adapting the support radius is common practice, so far, no approach aimed at using it to strictly limit the memory consumption of simulations.

Calculating and accessing particle neighborhoods are core problems of every SPH framework. To that end, the simulation domain is usually subdivided by uniform grids [HKK07; Gos+10; Gre10; OK12] often in combination with spatial hashing [Ihm+11] or by hierarchical data structures [HK89; Gon15] into which particles are sorted. These data structures allow particles to be accessed based on their physical location. Often cells are ordered by space filling curves and to increase cache efficiency particles are sorted accordingly so that particles that are close neighbors in memory are also close neighbors in simulation space [Gos+10; Ihm+11; Dom+11]. During simulations each particle has to traverse all possible neighbors in this data structure. Using hierarchical data structures is rather costly and thus usually only applied for compressible simulations with strongly varying support radii and gravitational codes [HK89]. For incompressible flows, uniform cells are commonly used with cell sizes of H^3. Then for each particle 27 cells have to be searched for neighbors. However, still about 87% of these potentially neighboring particles lie outside the particle's support radius and thus don't interact [DA12].

To prevent these spurious particle pairings, neighbor lists can be calculated which explicitly store all pairs of interacting particles [Ver67; Dom+11]. Especially in incompressible simulations using iterative solvers [Ihm+13; BK15], many particle interpolations have to be performed in each time step. Computing a neighbor list only once per time step instead of calculating all possible interactions for every interpolation strongly improves performance [Ihm+11; GEF15]. Neighbor lists can either be processed in two passes, a first pass to calculate the number of neighbors to allocate enough memory and a second pass to actually find the neighbors [VBC08], or by pre-allocating a fixed array with a maximum number of neighbors per particle [Dom+11].

The simulation of SPH-based fluids can be efficiently carried out on GPUs using regular grids to subdivide the simulation domain [HKK07; Gre10; Gos+10; GEF15]. As hierarchical data structures and hashing involve irregular access patterns and cause thread divergence, they are usually avoided. Due to the restricted amount of memory, particle neighborhoods are often accessed by traversing cells for each interpolation [HKK07; Gre10; Gos+10]. Explicitly storing neighbor lists on the GPU [OK12; GEF15] can get very memory-consuming and thus strongly limits the number of particles if the neighborhood size is unbounded. So far, no approach to restrict the size of neighborhoods has been presented.

2.3 Variable support SPH

There is an ideal number of neighbors N_H inherent to every kernel function which depends on the shape of the kernel. For the cubic spline kernel the ideal number of neighbors is given as $N_H = 50$ [DA12]. For our improved neighbor list algorithm we propose a formulation of the support distance that is derived based on the ideal number of neighbors.

The support distance H_i and the number of neighbors N_H are closely related as within a sphere of radius H_i only a certain number of particles with their respective volumes $\widetilde{V}_i$ can be found. This relation can be described as $N_H = \frac{\frac{4}{3}\pi H_i^3}{\widetilde{V}_i}$. Solving for H_i then yields our proposed formulation to locally adjust the support radius as

$$H_i = \widetilde{V}_i^{\frac{1}{3}} \underbrace{\left(\frac{N_H}{\frac{4}{3}\pi}\right)^{\frac{1}{3}}}_{s_h \eta} . \tag{2.3}$$

That way, we determine η depending on the properties of the kernel rather than by manually tweaking.

Our formulation for adapting the support radius is similar to the commonly used one, see Eq. 2.2. However, we use the adaptive particle volume of Solenthaler and Pajarola [SP08]

$$\widetilde{V}_i = \frac{1}{\sum_j W(x_{ij}, H)} \tag{2.4}$$

instead of $\frac{m_i}{\rho_i}$. In contrast to our general volume estimate, according to [Ort+13], we chose this formulation for the support distance as we are interested in the actual spatial distribution of the particles and not a corrected distribution that takes into account the particle volume. If the particle volume is taken into account, our estimate cannot adequately handle boundaries between particle resolutions as in those cases the adaptive volume would not significantly change due to its corrective effects. Equation 2.4 does not correct the summation for the different particle volumes and causes the desired change in the adaptive volume. For simulations of uniform particle rest volumes both formulations lead to the exact same result.

The proposed formulation locally adapts in two ways to the current simulation. On the one hand for surface particles without a full neighborhood our formulation increases the support distance and helps fill up the surface particle's neighborhood which can increase the SPH interpolation at the surface. On the other hand, for spatially compressed particles, we reduce the support distance and hence reduce the number of neighbors to meet the desired maximum.

In the first time step of a simulation, we can only assume an initial distribution where $\widetilde{V}_i = V_i$ as we cannot calculate the actual estimate without having a support distance. As we can assume temporal coherence of the particle neighborhood, we later use the support radius of the previous iteration to give better approximations for $\widetilde{V}_i$. To facilitate this we apply a linear interpolation of the newly predicted support radius H_i using Eq. 2.3 and the support radius of the previous timestep H_i^l as

$$H_i^{l+1} = H_i^l + \omega \left[H_i - H_i^l \right] , \tag{2.5}$$

where ω is the weight which we usually set as $\omega = 0.5$. By exploiting temporal coherence, we often find an estimate of the support radius that directly yields the desired neighborhood size.

2.4 Constrained neighborhoods

Our proposed constrained neighborhood algorithm aims at efficiently calculating particle neighborhoods of a fixed, user-defined size. Using a fixed size allows giving strict bounds on the memory consumption and also allows us to calculate the neighbor list in a single pass over the underlying particle access data structure in most situations. Our algorithm works in an iterative way similar to a prediction-correction method using a soft start by initializing the support radius according to Sec. 2.3 and then predicting a neighborhood using this support distance and correcting potential errors. Note that the underlying data structure of our neighbor lists is an array. We only use the term list to be consistent with literature.

While our initial support radius gives good estimates, it is not able to guarantee neighbor limits as we always get a distribution of neighborhood sizes above and below the desired number N_H. Although our algorithm is able to enforce a strict limit of N_H neighbors, in some scenarios it can be beneficial to allow each particle to store N_{add} neighbors, so that in total each particle can have $N = N_H + N_{\text{add}}$ neighbors.

Fig. 2.2 shows a flow diagram of the proposed algorithm. In the iterative process we first try to create a neighbor list of fixed size (see Sec. 2.4.1). While adding neighbors to the list, we store the support radii of the farthest particles, i.e. the fringe, of the neighborhood in a *fringe buffer* of limited size and if necessary, additional particles are stored in an *overflow buffer* (see Sec. 2.4.2). If too many neighbors are found, the support radius is reduced using the fringe buffer. As neighboring support radii may also have been reduced, we try to replace corresponding neighbors with neighbors from the overflow list (see Sec. 2.4.3). Only if we run out of space in the overflow list, or cannot reduce the support radius sufficiently in a single iteration due to resource constraints, we have to do another iteration over the particle access data structure. Due to the soft start and the overflow buffer, in most simulation time steps the neighborhood can be calculated in a single iteration consisting of a single pass over the underlying data structure.

2.4.1 Initial neighborhood list

First, we calculate a neighbor list of at most N elements for each particle i using a single pass over the underlying data structure. The fixed space of the neighbor list, however, may not be large enough to directly contain all neighbors inside the support radius H_i.

In case, no particle finds more than N neighbors, we are done and the predicted support radius of Sec. 2.3 is valid for all particles. This corresponds to the green control flow in Fig. 2.2. In case any particle finds more neighbors, the respective support radius has to be decreased in order to reduce the number of neighbors and fit the strict memory bounds. In the worst case, additional iterations of the neighbor search have to be run.

2.4.2 Overflow lists and tracking the neighborhood fringe

In order to prevent additional iterations of the neighbor search, we keep track of neighbors that initially do not fit into the neighbor list. Therefore, we use an

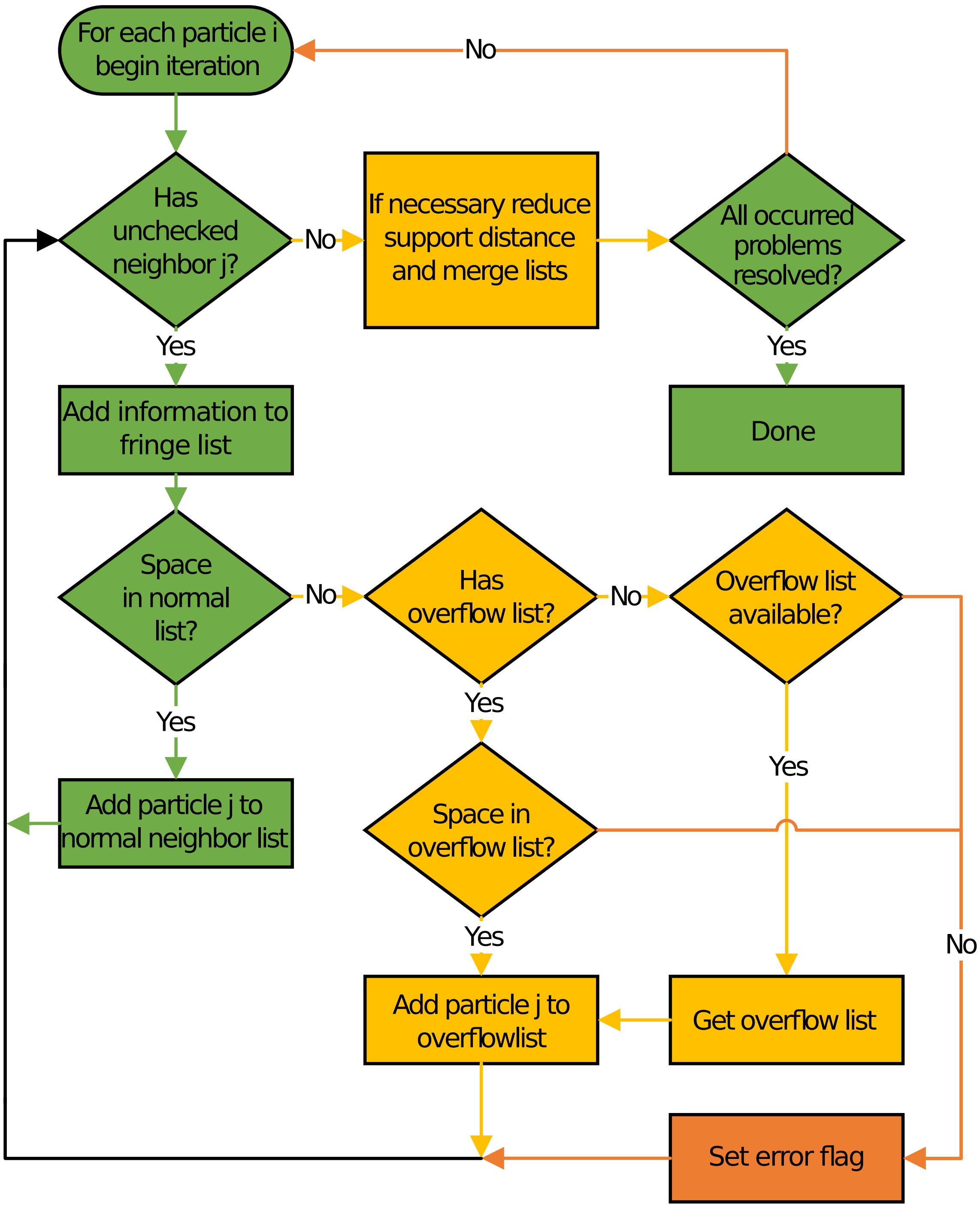

For each particle i begin iteration
No
Has unchecked neighbor j?
No
If necessary reduce support distance and merge lists
All occurred problems resolved?
Yes
Yes
Add information to fringe list
Done
Space in normal list?
No
Has overflow list?
No
Overflow list available?
Yes
Yes
Yes
Add particle j to normal neighbor list
Space in overflow list?
Yes
No
Add particle j to overflowlist
Get overflow list
Set error flag

Figure 2.2: Flow chart of our proposed neighbor algorithm. Green elements describe the path of the method that is always done. Yellow elements describe our added functions to handle the overflow and to correct the support distance. If the error flag is set (red box), our algorithm needs another iteration as it cannot accurately handle all errors in this iteration.

overflow list that is subdivided into fixed-size partitions. In this list we can store a limited number of indices of additional neighbors that did not fit into the normal list without correction. For each particle we can only grab one partition and, in order to keep memory consumption low, the total number of partitions is limited.

During the construction of the initial neighbor list, we already keep track of the outermost neighbor particles j for each particle i in the fringe of the neighborhood, i.e. the farthest particles from i. To that end, we propose to use a *fringe buffer* that stores the largest possible support radii $H_i = 2x_{ij} - H_j$ for particle i so that the distance x_{ij} to the neighboring particle j is just inside the symmetrized support $\frac{H_i+H_j}{2} = x_{ij}$. For each particle the fringe buffer holds the same number of elements as a partition of the overflow list, however, the fringe buffer is stored in shared memory in order to allow for an efficient sorting of its elements. We only allow to store unique values in the fringe buffer and only replace the current minimum value if a value is found that is larger than the current minimum. Although it only occurs very rarely, if two or more elements had exactly the same value, the algorithm might not terminate otherwise. We split the storage of distances and indices as we need to sort the distances and replace values within the fringe buffer but only add values to the overflow list, see Fig. 2.2.

If a particle i has found M neighbors with $M > N$, we have to correct the support radius so that only N particles remain inside H_i. In order to find the proper support radius, we use the fringe buffer and sort it in descending order by distance. The $(M - N)$-th entry in the fringe buffer is then used to directly set the support radius of particle i as this is the support radius that particle i needs to have in order to just contain the targeted neighbor number. Note that as we only store unique values in the fringe buffer, it may rarely occur that the support radius is slightly over-corrected.

2.4.3 Merging of initial neighborhood and overflow list

If a particle has used an overflow partition, its support radius has been reduced in the previous step. As neighboring particles are likely to have found too many neighbors, too, these possibly can be removed from the neighborhood as well. Additionally, we want to merge the entries in the overflow list with the normal list.

In order to remove these particles, we simply iterate over the neighbor list to find particles that are now outside the support radius. These are replaced by particles from the overflow list that are within the support distance. That way, we effectively merge the initial neighbor list with the overflow list. Only if particles remain inside the support radius that do not fit into the neighbor list, we have to run another iteration to further correct the support radius.

To keep this process efficient we only apply it to particles with an overflow list and not for every particle in the simulation. Although this causes some spurious particle interactions, the impact on the simulation performance can be neglected due to the very limited number of spurious interactions. As we use symmetrized support radii in the SPH integration, particle interactions always remain symmetric so that conservation of momentum is guaranteed.

2.4.4 Discussion

A basic approach to strictly limit the size of the neighborhood is to use a constant support radius but only search up to N neighbors for each particle and then stop. This, however, causes simulations to get highly unstable because, on the one hand, the particle neighborhood is no longer symmetric and, on the other hand, the kernel's normalization property is violated if particles are removed from the neighborhood without properly adjusting the support radius.

In contrast, our proposed algorithm works by first predicting and then iteratively correcting the support radius in order to reduce the number of neighbors. As only the farthest particles of the neighborhood get removed, no instabilities due to asymmetric interactions occur and due to the fact that the support radius is properly adjusted, the normalization property of the kernel function is still satisfied. The support radius is reduced iteratively using a value from the fringe buffer. As the fringe buffer only contains unique values that are smaller than the current support radius, the support radius can only be reduced by our algorithm. Hence, the size of the neighborhood will also be reduced in each iteration and the algorithm terminates successfully.

2.5 Implementation details

2.5.1 Overflow lists

Although in most simulation scenarios only few particles find more neighbors than the user-defined limit, this would always cause a second iteration of the neighbor search. In order to prevent additional iterations, we store the overflow of the neighborhood lists in an additional buffer (see Sec. 2.4.2) and later try to merge entries with the neighbor list. We set the size of this buffer so that each particle can have one additional neighbor. Each partition of the buffer has 12 elements which corresponds to the size of the fringe buffer that we only store in shared memory. We use a single counter in global memory that points to the first free partition in the buffer. If a particle needs an overflow partition, we use the current counter value as start address and increment the counter using an atomic-add operation. If more overflow lists than available are requested, an error flag is set and we have to start another iteration.

2.5.2 Constraining the initial support radius

In our implementation, we use a cell based particle access data structure with uniform cell sizes. The cell size limits the support radius that is allowed for a particle without clipping the neighborhood. Initially, the cell length is set to fit a particle with no compression (see Eq. 2.3) and we later limit support radii to this length.

2.5.3 Coalesced neighbor lists

One technical improvement, we propose, is the use of a coalesced neighbor list. In traditional neighbor lists [OK12], every particle stores its neighbors contiguously in memory. This is a very straight forward implementation, however, on GPUs this approach does not perform well due to non-coalesced loads and small cache sizes per particle. In order to resolve this issue, we propose an optimized data structure in which the classical neighbor list is stored in transposed order. So we contiguously store the first neighbor of every particle and then the second neighbor of every particle and so on. This optimization allows us to access the neighbor list in fully coalescing loads without cache problems.

Note, this structure is only realistically possible if the length of the neighbor list for each particle is strictly limited. Otherwise, the size of the data structure would have to be adjusted to the particle with the most neighbors which would be highly restrictive to simulating large particle numbers.

2.6 Results and discussion

In order to evaluate the runtime performance and memory consumption and in order to show the versatility and stability of our approach, we ran several test cases. To enforce fluid incompressibility, we used IISPH [Ihm+13] and DFSPH [BK15] resulting in an average compression rate of 0.5%. Dynamic rigid objects were realized using particles [Aki+12]. For static boundaries we used distance fields [HKK07]. We used the model of Akinci et al. [AAT13] to simulate surface tension effects. Simulations were run on an Intel i7-5930K with 16GB of host RAM and an Nvidia GeForce Titan X with 12GB of device RAM. Fluid surfaces were rendered using screen-space curvature smoothing [LGS09].

First, we present the capabilities of our approach in real-time simulations. Then, we will give a comparison to previous neighbor algorithms in terms of memory consumption and computational speed and present simulations of very large scale. In order to demonstrate the stability of our method, we show complex scenarios of multi-scale simulations, rigid-fluid coupling and test the compatibility with different kernel functions. Finally in order to demonstrate the scaling and performance of our method we test our approach against previous approaches for a dambreak scenario using various resolutions.

2.6.1 Real-time simulations

For real-time simulations, at least 30 frames have to be simulated per second. With our improved algorithm we were able to run a real-time simulation of a dam break scenario with over 500K particles using DFSPH and the poly6 kernel, see Fig. 2.1 (left). Using the cubic spline kernel, still 240K particles were simulated in real-time.

2.6.2 Scaling and comparison to previous work

We compared our new method to three previous approaches by simulating a bunny that was sampled with fluid particles and dropping it into a basin (see Fig. 2.3). The simulation ran with 2.4M particles for 15s simulated time using IISPH with cubic spline kernel. Table 2.1 gives the performance results averaged over all time steps.

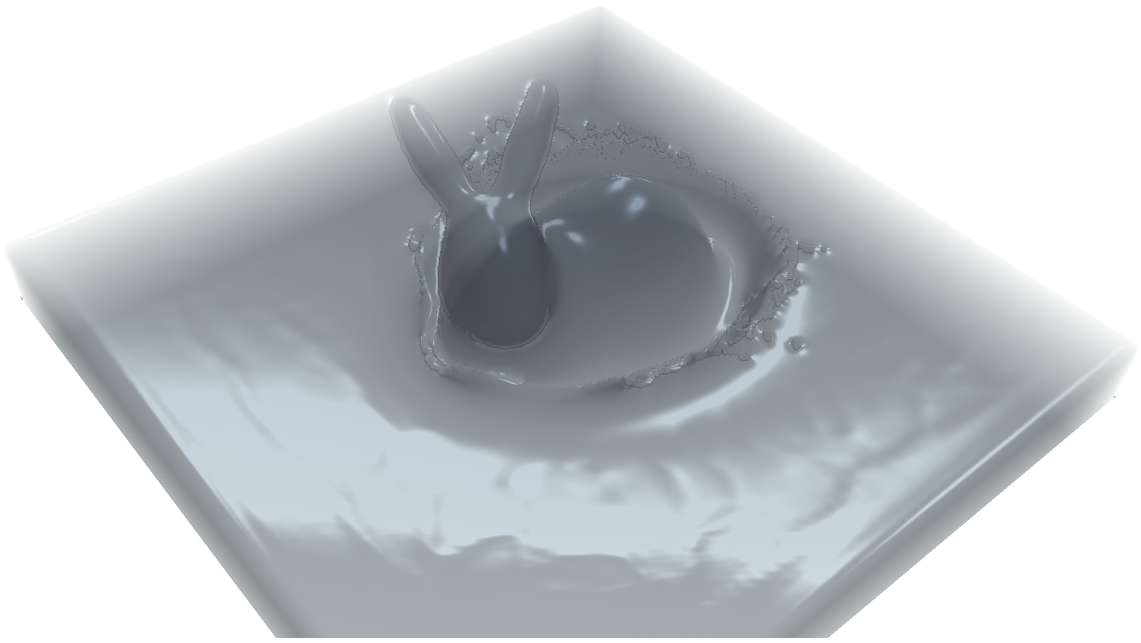


Figure 2.3: A bunny is sampled with fluid particles and dropped into a basin. Surfaces are rendered using curvature smoothing.

Table 2.1: Performance characteristics of the Bunny scene for different neighbor algorithms. 'Frametime' gives the run time of a simulation time step. 'Corrected' gives the number of particles for which our algorithm adjusted the support radius. 'Ratio' gives the time spent to calculate the neighbor list and sort the particles vs. the total time. 'Size' is the neighbor list size in device memory.

Algorithm	**Frametime (ms)**	**Corrected**	**Ratio**	**Size (MB)**
Our approach $N = N_H = 50$	527	51923	9.07%	475
Constant H, coalesced	596.39	-	8.40%	932.61
Two-pass [OK12]	1323.39	-	3.33%	699.457
Cell iteration [Gre10]	765.83	-	1.2%	-

We ran our approach with $N = 50$ and outperformed all other approaches in simulation speed and neighbor list based approaches in memory consumption. The 'Two-pass' algorithm calculates the neighbor list in two passes over the particle access data structure and stores the exact number of neighbors for each particle contiguously in memory [OK12]. In comparison, our algorithm reduced the memory consumption by 47%. Figure 2.4 shows the memory consumption of the neighborhood sizes resulting from using constant support radius and our adaptive support radius. Most particles in the fluid volume have more than N neighbors using constant support. Our method, however, is able to strictly enforce the desired number of neighbors. The row 'Cell iteration' corresponds to the approach in which the particles neighborhood is recomputed for every interpolation [Gre10]. It does not need any memory to store neighbor lists but took over 45% longer than our proposed approach to simulate. The row 'Constant H, coalesced' gives the results for a neighbor list approach with fixed size that uses our proposed coalesced data structure. By strictly limiting the neighborhood size, we were able to reduce

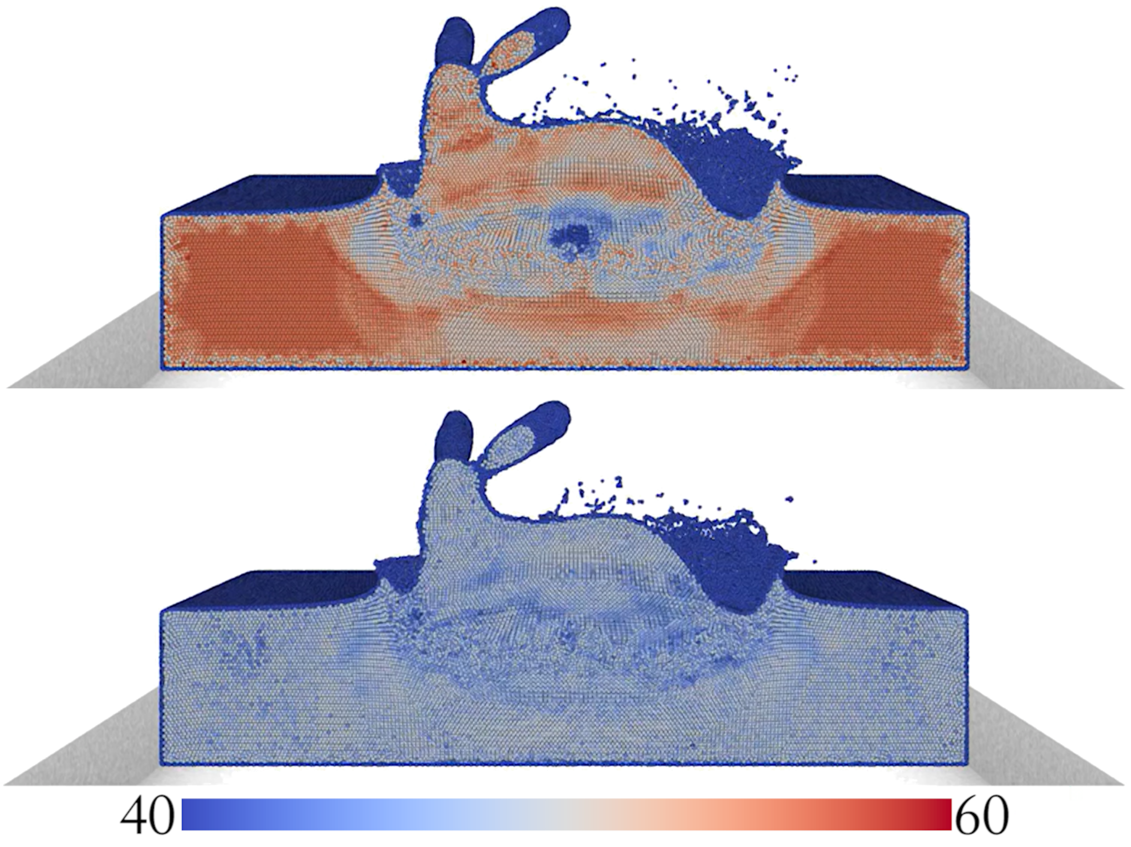


Figure 2.4: Comparison of the Bunny scene using constant support radius and our method using $N = 50$. The number of neighbors is color coded.

the computation time significantly whilst consuming less memory than previous neighborlist based approaches.

In order to evaluate the scaling of our method in comparison to previous works, we simulated 15 seconds of a dambreak scenario (see Fig. 2.9) using various particle resolutions yielding simulations of 30K to 4M particles. We compared our algorithm using different neighborhood sizes $N = 50, 55, 60$ with the naïve neighbor list approach (Two-pass [OK12]), a cell based approach (Cell iteration), and an approach using our proposed coalesced data structure without constraints (Coalesced). Fig. 2.5 shows the average overall frame- time of the different approaches for different particle resolutions. Our method was able to handle small and large simulations very well and outperformed the previous approaches for all particle resolutions in terms of the overall frametime. This is due to the fact that our method creates a lower average number of neighbors than previous approaches, as shown in Fig. 2.6, so that the costly time integration of the SPH equations is sped up considerably whilst using an efficient access structure. On average our approach found about 10% less neighbors than methods using a constant support radius for the cubic spline kernel, when using strict constraints of N=50 neighbors, without negatively influencing the simulation behavior. The smaller N is chosen, the smaller the average number of neighbors gets. Fig. 2.7 shows the ratio of the SPH integration and the overall frametime, the remaining time is spent for the neighborhood algorithm. With our approach relatively more time is spent for

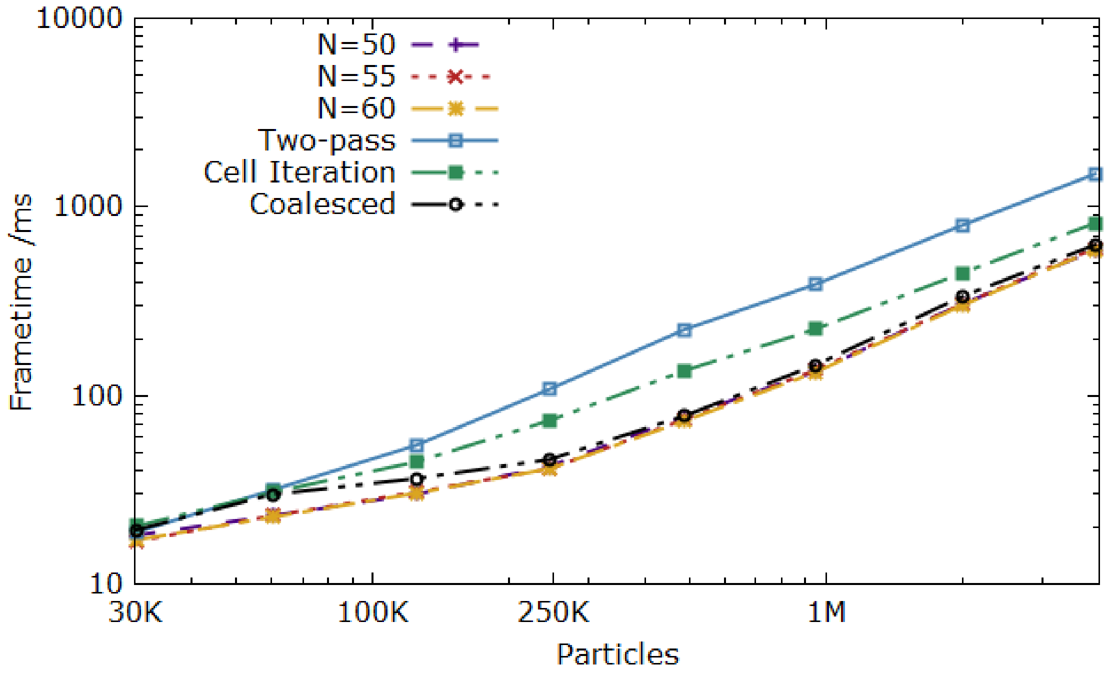


Figure 2.5: Average time to simulate one frame (y-axis) for different particle resolutions (x-axis) of our neighbor algorithm (N=50, 55, 60) and previous works.

finding neighboring particles than for the integration. Although our method introduces a larger computational overhead over previous approaches, it yields smaller neighborhood sizes and hence considerably reduces the time spent for the time integration and is able to speed up the overall simulation time.

2.6.3 Large scale simulations

One of the main benefits of the proposed approach is that more particles than with previous neighbor list approaches fit into memory. A large scale scenario with up to 35M particles in which a stream of water is perturbed by two pillars could be simulated, see Fig. 2.1 (right). The simulation took 3.5 s per simulation step and the neighbor list consumed a total of 2240MB with a strict limit of $N = N_H = 15$ neighbors using IISPH and the poly6 kernel. The only other method able to simulate 35M particles was the cell iteration. However, it took about 6.8s to simulate a time step.

2.6.4 Stability and versatility of our approach

Without additional adjustments, our method was able to stably simulate multi-scale scenarios with volume ratios of 1:8 using $N_{\text{add}} = 25$ additional neighbors (see Fig. 2.8). We were also able to handle volume differences with a ratio of 1:2 without additional neighbors required. We also tried to run simulations in which we stopped adding neighbors to the neighbor list after $N = N_H$ neighbors had been found. This however caused severe instabilities due to asymmetric interactions. We thus omit to show these simulations.

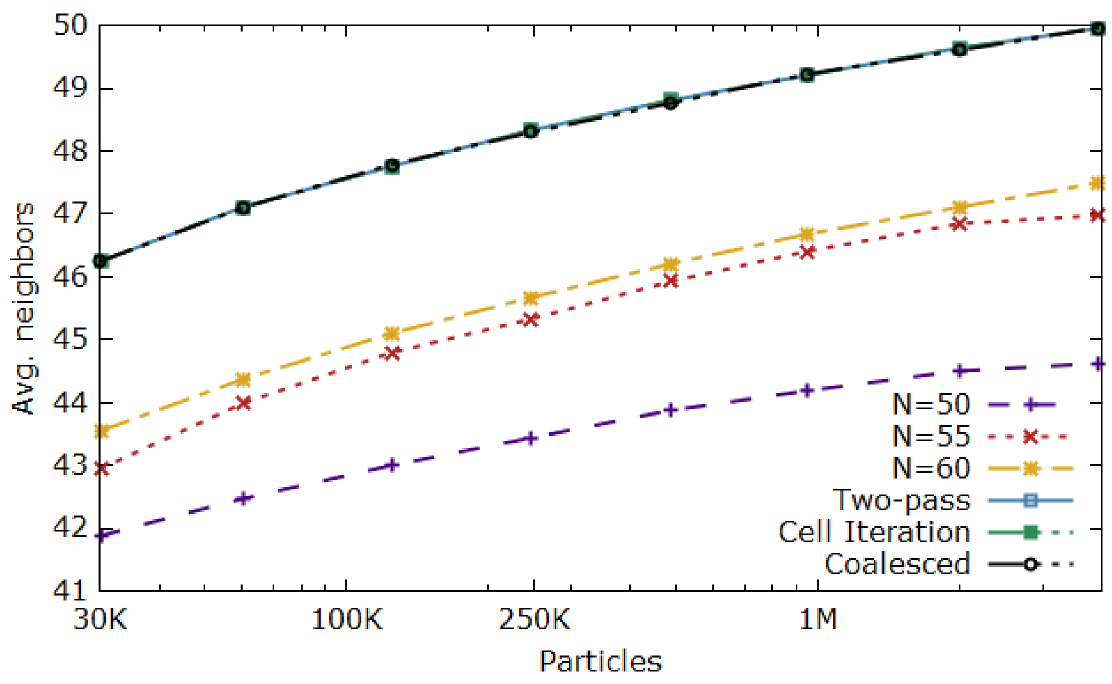


Figure 2.6: Average number of neighbors per particle (y-axis) for different particle resolutions (x-axis) of our neighbor algorithm (N=50, 55, 60) and previous works. Note that Two-pass, Cell iteration and Coalesced do not adjust support radii and thus all yield the same number of neighbors.

Additionally, our method works with two-way rigid-fluid coupling (see Fig. 2.10) where the fluid particles are directly influenced by dynamic rigid particles.

During our experiments, we also evaluated the effect of different N_{add} for single-scale simulations using the cubic spline kernel. It showed that only in combination with multi-scale simulations additional neighbors were necessary to achieve stable simulations. Both for run time performance and memory consumption, $N_{\text{add}} = 0$ yielded the best results.

Although for interactive simulations poly6 or cubic spline kernels are usually employed, more stable kernels are commonly used in scientific computing [DA12]. We were able to stably simulate the Dam Break scenario of Fig 2.9 using seven different kernel functions (poly6, cubic spline, quartic spline, quintic spline, Wendland2, Wendland4, Wendland6) and set $N = N_H$ to their respective ideal neighborhood size [DA12].

When using variable support radii, both the particle positions and support radii are time-dependent. Although, in general, this has an influence on the derivation of time-derivatives, we omitted to take derivatives of H_i into account as their effects have been shown to be negligible [HK89] and we did not observe any issues concerning simulation stability by ignoring these terms.

2.7 Conclusions

In this paper, we presented a novel algorithm to efficiently calculate memory-constrained neighbor lists for SPH-based particle simulations. The approach

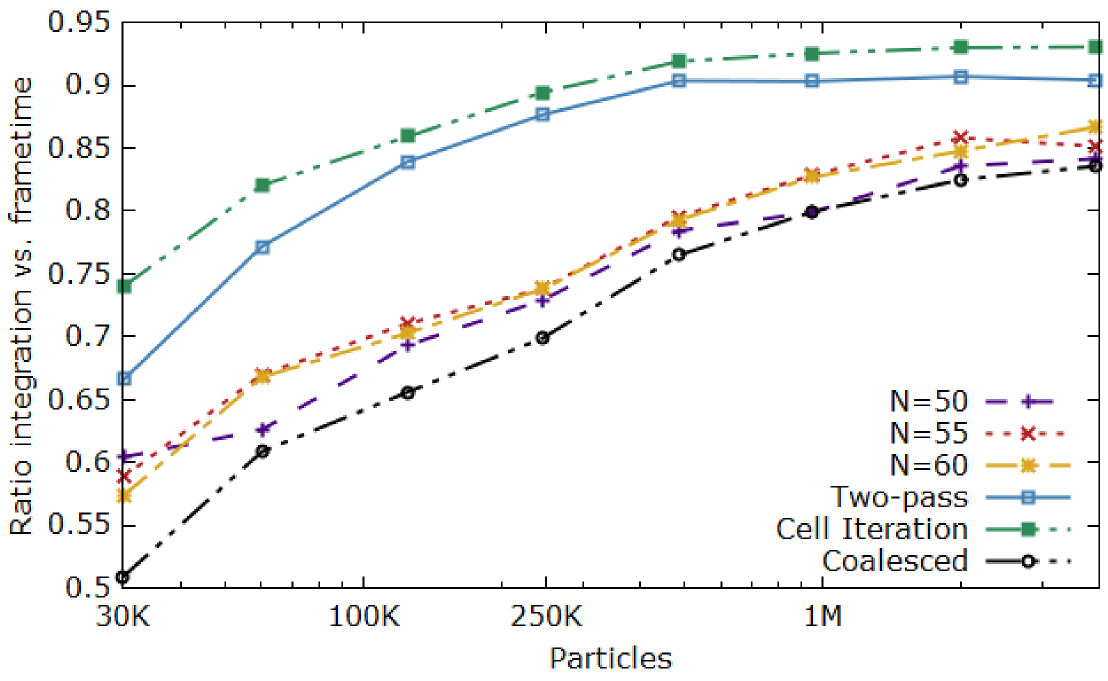


Figure 2.7: Ratio of the time spent in the integration part of the simulation step over the overall simulation frametime (y-axis) for different particle resolutions (x-axis).

works by iteratively adapting each particle's support radius so that a user-defined maximum number of neighbors is never exceeded. The algorithm works iteratively in a predictive-corrective way, where the proposed initial prediction is based on the current simulation state. In order to improve performance, an optimized data structure for neighbor lists has been proposed that allows for fully coalesced memory reads on GPUs.

Due to the restricted neighborhood size, both performance and memory consumption can be considerably improved compared to previous approaches. We are able to stably simulate incompressible fluids including two-way fluid-rigid coupling and multi-scale simulations. Due to our highly efficient method, over 500K particles could be simulated in real-time and, due to the strictly limited memory consumption, up to 35M particles could be simulated on a single consumer GPU.

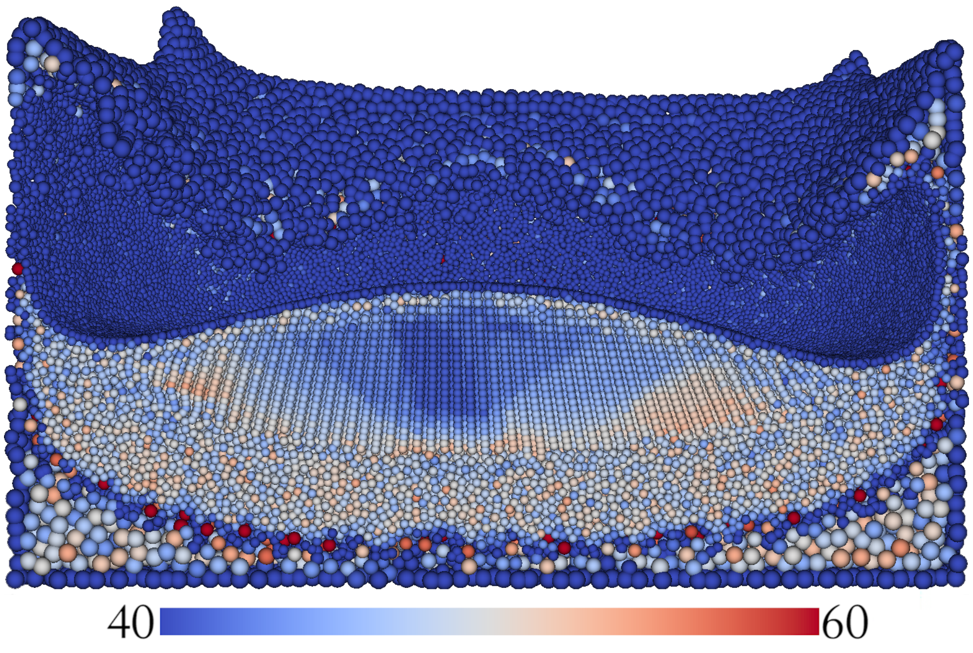


Figure 2.8: A spherical drop of particles with $r = 0.5m$ drips into a fluid volume with $r = 1.0m$. We set $N = N_H + 25$ to handle the boundary between particle resolutions. For an unbounded neighbor list we set $N = N_H + 150$. The number of neighbors is color coded.

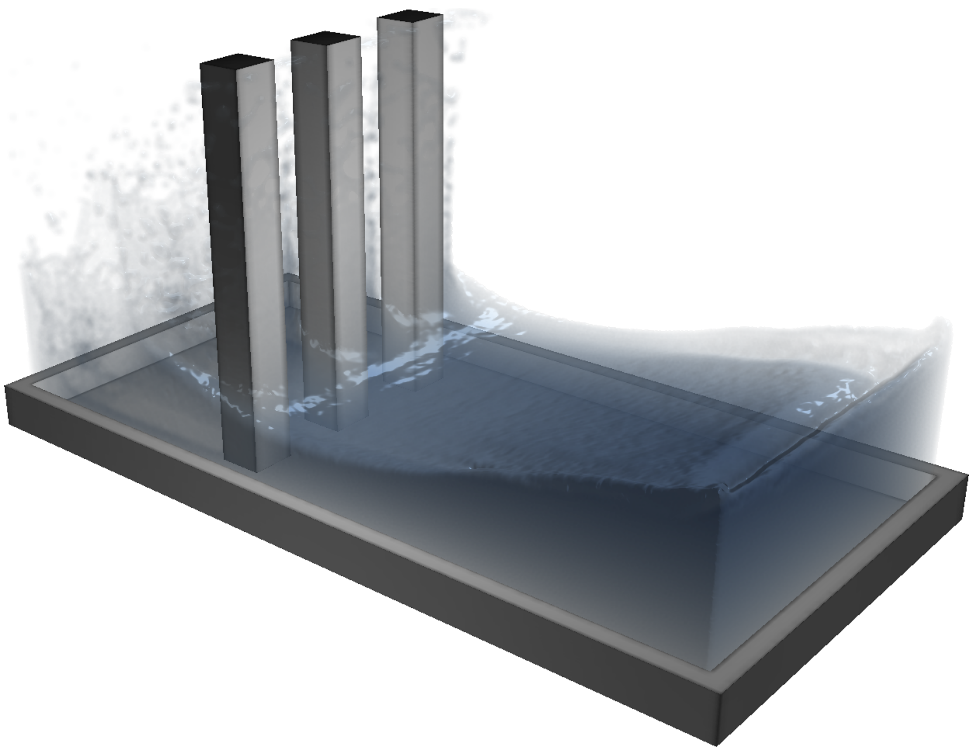


Figure 2.9: Dam break scenario with 500K particles. Surfaces are rendered using curvature smoothing.

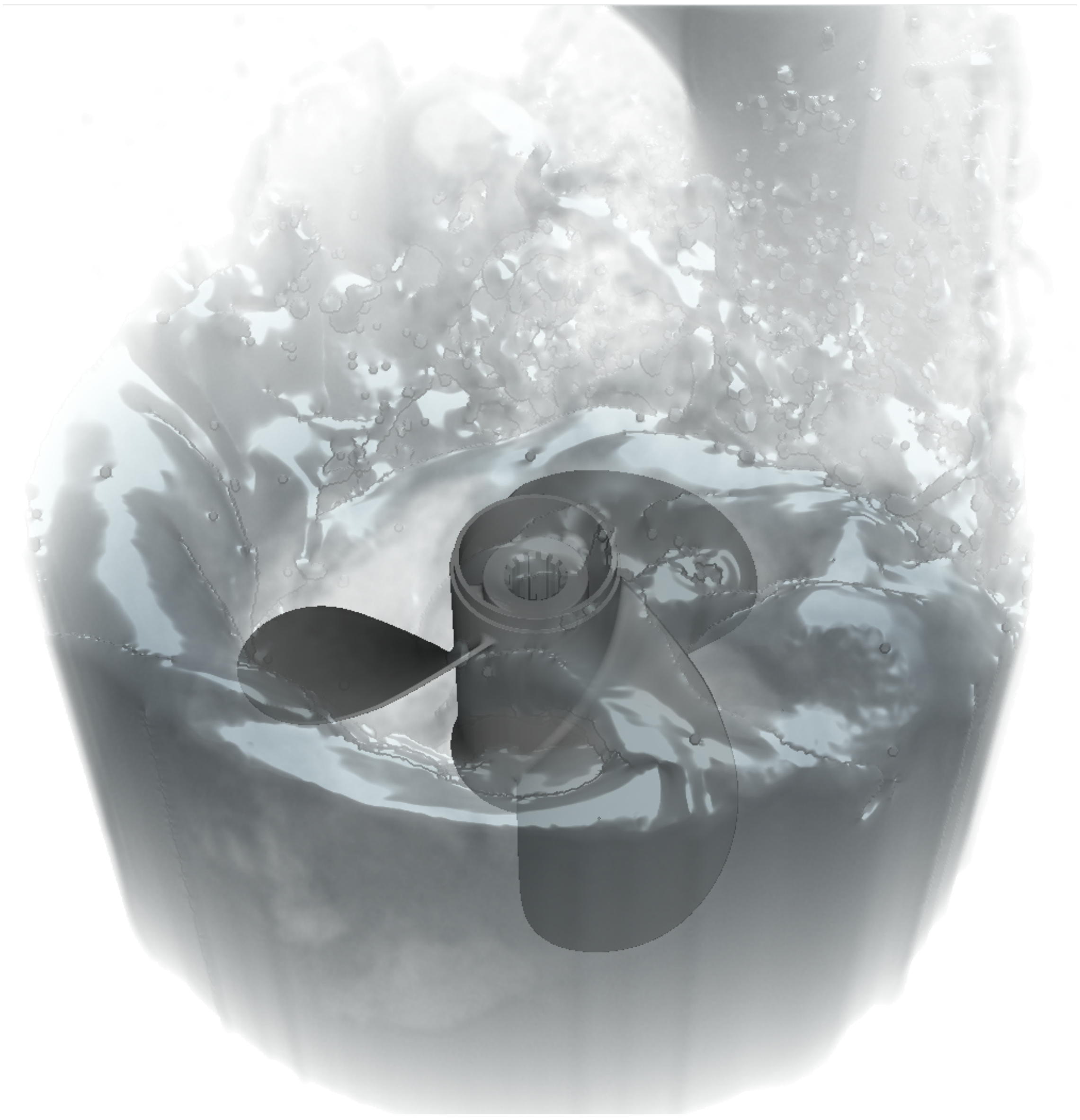


Figure 2.10: A mixer is filled with 1.3M particles. The spinning rotor is simulated using dynamic rigid particles. The neighborhood size was set to $N = N_H$. Surfaces were rendered using curvature smoothing.

Chapter 3

Infinite Continuous Adaptivity for Incompressible SPH

Contextualization

This chapter reprints the publication “Infinite Continuous Adaptivity for Incompressible SPH”[1] published as a conference proceeding of SIGGRAPH 2017 via ACM Transactions on Graphics [WHK17]. This work represents the seminal work regarding to spatial adaptivity on which all further research was based, and introduced the concept of sharing to optimize spatial resolution gradients and multiple different refinement patterns per particle and the implicit blending approach.

Principally, the focus of this paper is on introducing an overarching spatially adaptive and incompressible SPH simulation into a computer animation context. This is achieved through the introduction of a continuous sizing function $S(\mathbf{x})$, which dictates the ideal resolution at a given spatial location $\mathbf{x}$, where particle splitting, particle merging and mass sharing are used to minimize the difference of actual particle sizes with respect to their ideal size. Furthermore, this paper reintroduces temporal blending, which was previously proposed by Orthmann and Kolb [OK12], in an implicit manner, i.e., without tracking actual parent particles. This paper uses the constrained neighborlists approach, see Chapter 2, to limit the number of neighbors of particles in regions with large resolution gradients to reduce numerical instabilities. Finally, this paper uses multiple refinement patterns, i.e., a single particle is replaced with 2 to 16 particles, to smoothen resolution gradients and to not split the same particle multiple times in quick succession.

The initial motivation for this paper arose during the evaluation of the prior paper [WHK16], which made spatial adaptivity computationally feasible, but required density errors from refinement to be reduced. The core ideas of the sizing function, implicit blending and the construction of refinement patterns came from Rene Winchenbach, with input from Hendrik Hochstetter on the design of the refinement patterns and their optimization. Furthermore, Hendrik Hochstetter and Andreas Kolb provided significant feedback and assistance during the writing process of the paper. Finally, the anonymous reviewers of the SIGGRAPH 2017 conference provided valuable feedback and inspiration for further research opportunities.

[1]Note that this paper denotes adaptivity ratios as $1:n$, i.e., 1 low resolution particle is equivalent to n high resolution particles, whereas most other papers denote adaptivity ratios as $n:1$.

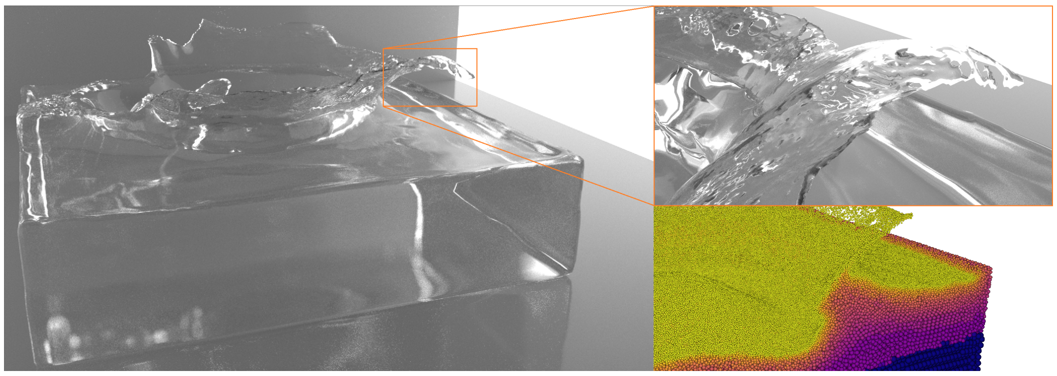


Figure 3.1: Adaptive simulation of a bunny shaped drop of water falling into a tank at particle mass ratios of 1:300. The left image shows the overall splashing effect while the top right image shows the fine visible surface detail. The color mapped image shows a cut away view with volume color coded from purple to yellow. Our algorithm allows for a smooth adaptive simulation with detailed surface and low interior resolution.

Abstract

In this paper we introduce a novel method to adaptive incompressible SPH simulations. Instead of using a scheme with a number of fixed particle sizes or levels, our approach allows continuous particle sizes. This enables us to define optimal particle masses with respect to, e.g.,, the distance to the fluid's surface. A required change in mass due to the dynamics of the fluid is properly and stably handled by our scheme of mass redistribution. This includes temporally smooth changes in particle masses as well as sudden mass variations in regions of high flow dynamics. Our approach guarantees low spatial variations in particle size, which is a core property in order to achieve large adaptivity ratios for incompressible fluid simulations. Conceptually, our approach allows for infinite continuous adaptivity, practically we achieved adaptivity ratios up to 5 orders of magnitude, while still being mass preserving and numerically stable, yielding unprecedented vivid surface detail at comparably low computational cost and moderate particle counts.

3.1 Introduction

Fluid simulation has been a topic of interest for a long time and has found widespread use in computer animation. While the plausibility and vividness of simulated fluids strongly depend on the dynamics in specific regions, e.g.,, at the fluid surface including droplets, splashing and formation of fluid sheets, large parts of the fluid bulk are less important for the overall visual appearance of the simulation. Using a high resolution in specific flow regions, like surfaces, and a coarse resolution in other parts, like the bulk, has a huge potential to improve efficiency at no or minimal loss of visual quality.

For grid-based fluid simulation there are various methods to achieve adaptivity, e.g.,using octrees [LGF04], non-uniform [Kli+06] or tetrahedral meshes [ATW13]. Grid-free, particle-based approaches, such as *Smoothed Particle Hydrodynamics*

(SPH), have quite some advantages over grid-based approaches with respect to mass preservation and modelling of free surfaces. However, only a limited amount of adaptivity has been achieved for SPH-based simulations of incompressible fluids so far. All existing methods simulate particles on a pre-defined set of *discrete particle levels*, each fixating a pre-defined particle size [Ada+07; SG11; OK12; HS13]. Depending on the refinement requirements at a given spatial location, a specific level, i.e.,, particle size, is chosen. If the currently used particle size needs to be altered, particles can be replaced instantaneously [Ada+07] or by applying temporal blending schemes [OK12]. Alternatively, higher resolution can be achieved by simulating separate scales in regions which require higher resolution which either suffer from mass loss [SG11; HS13] or only allow for very limited adaptivity [Cor+14].

Indirect coupling between particle levels is either limited to small particle mass ratios of 1:8 [Cor+14] or is not mass preserving [SG11; HS13]. Level-based incompressible fluid simulations cannot ensure sufficiently smooth transitions between regions of different particle resolutions. Thus, direct interaction of particles of different levels can cause instabilities which so far restrict adaptivity to particle mass ratios of up to 1:64 [OK12].

In this paper we introduce the concept of adaptivity using arbitrary particle sizes. The main idea is to have a *continuously adaptive mass* for each particle, which is driven by the distance to important flow regions, e.g.,, the fluid surface. Our approach enforces a smooth transition between areas of different resolutions by directly exchanging mass between particles. As a consequence, our technique does neither suffer from mass loss like approaches with indirect coupling between different resolutions, nor from numerical instabilities due to the interaction of particles with very different sizes and masses. Thus, we allow for virtually *infinite adaptivity*.

Conceptually and technically our approach comprises the following features and contributions:

- the concept of continuously adjusted particle masses (and sizes) with restricted spatial variation, leading to technically unlimited adaptivity,
- a method to flexibly adjust particle masses not only by split/merge operations but also by redistributing mass among particles which guarantees mass preservation and prevents problems of finding merge partners,
- an efficient implicit temporal blending method which allows for stable incompressible fluid simulation, and
- proper handling of continuously varying particle radii in the SPH simulation framework.

The rest of paper is structured as follows. First we will discuss related work in Sec. 3.2 and recapitulate the foundations of SPH in Sec. 3.3. Sec. 3.4 gives an overview of our approach. Sec. 3.5 describes how to calculate a particle's distance to the surface and its desired optimal size. Our novel splitting and merging schemes and our smooth and conservative mass redistribution are described in Sec. 3.6. In Sec. 3.7 we discuss our novel concept of implicit temporal blending. Sec. 3.8 shows how to adjust the simulation to stably work with varying particle resolution. Sec. 3.9 presents results before conclusions are drawn in Sec. 3.10.

3.2 Related work

Since its introduction by Gingold and Monaghan [GM77], SPH has been a very active field of research. First, stiff equations of state were employed to achieve weakly compressible fluids [MCG03; BT07]. Later, prediction-correction-based [SP09], iterative [MM13] and implicit methods were used to enforce incompressibility [Ihm+13; BK15].

Adaptive simulation using splitting and merging of particles was introduced by Desbrun and Cani [DC99]. Adams et al. [Ada+07] adjust particle positions after splitting to reduce the errors mainly introduced in the pressure term. As direct interactions between particles of different levels are a common source of instable simulations, Keiser et al. [Kei+06] use particles that also carry virtual particles of neighboring resolutions which are then used in the interaction. However, none of these approaches could be shown to work with incompressible fluids. A way to work around these instabilities is to use parallel separate simulation scales in which particles of different sizes interact only indirectly through coupling forces [SG11; HS13]. Although these methods work well with incompressible fluids, the coupling between resolution levels is rather unphysical and mass preservation is not guaranteed, as either the overall volume is not considered in the creation of particles [SG11] or the creation and deletion of high-resolution particles depend on random values [HS13]. Orthmann and Kolb [OK12] introduced a different approach to incompressible adaptive SPH by tracking the original particles after splitting and merging for a while and temporally blending the values of both resolutions to prevent abrupt changes in quantity fields. Although this allowed for stable simulations, it complicates the evaluation of quantities and the approach has only been shown to work with iterative pressure solvers and would require very expensive adjustments if used with current implicit solvers. Recently, Cornelis et al. [Cor+14] introduced IISPH-FLIP in which the pressure is solved for low resolution SPH particles while fluid advection uses smaller FLIP particles. Although the approach works mass-preservingly, it is restricted to mass ratios of 1:8 between SPH and FLIP particles.

In adaptive approaches with direct particle interactions, instabilities are mainly due to interactions of particles of very different sizes which currently limit adaptivity to mass ratios of 1:64 between the finest and coarsest resolution [Ada+07; OK12]. In order to improve the stability, different geometric shapes for splitting have been proposed. While Desbrun and Cani [DC99] found static 1:7-splitting to be stable, Adams et al. [Ada+07] and Orthmann and Kolb [OK12] used 1:2-patterns that are either dynamically optimized or temporally blended. In general, fix geometric refinement patterns can still cause large errors, thus, refinement patterns have been subject to optimization [Vac+13; Vac+16].

All prior approaches to adaptive SPH simulations are based on some notion of particle level, i.e.„ particles belong to one level of constant particle size and transitions between levels are achieved using fixed, optimized and/or blended $1{:}n$ refinement (split) and symmetric $n{:}1$ coalescing (merge) operations, preventing smooth adjustments of particle sizes. A crucial problem in the merge operation is the identification of *merge partners*, as only particles of the same level can be merged. Practically, particles often can not be merged due to lacking merge partners, which leads to isolated small particles in regions of coarse resolution and, thus, to massive numerical instabilities. Our approach of continuously varying particle masses and sizes solves all the above mentioned problems with adaptive, incompressible SPH-based fluid simulations. We guarantee smooth spatial transitions in particles sizes and, thus, we achieve numerical stability at unlimited adaptivity ratios.

3.3 SPH foundations

In SPH, quantities are interpolated from a weighted average of the surrounding particles' quantities as [Mon05]

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} W_{ij}, \tag{3.1}$$

where $A_i = A(\boldsymbol{x}_i)$ is the interpolated quantity for particle i at position $\boldsymbol{x}_i$ and j are the neighboring particles with quantities A_j. m_j and ρ_j denote the mass and density. $W_{ij} = W(x_{ij}, h_{ij})$ is the kernel function that weights contributions of the neighboring particles according to their distance $x_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ and their support radii h_i and h_j where the support radius is one of the most important factors in SPH. The support radius decides which particles interact how strongly with each other and is typically [Mon05] calculated for each particle i as

$$h_i = \eta \left(\frac{m_i}{\rho_i} \right)^{\frac{1}{3}}. \tag{3.2}$$

If two particles with different support radii interact, the support of either particle i can be used, i.e.,$h_{ij} := h_i$ (gather formulation), the support of particle j , i.e.,$h_{ij} := h_j$ (scatter formulation) or an average of both which yields a symmetric formulation $h_{ij} := \frac{h_i + h_j}{2}$ that is usually preferred. Throughout the text, we will assume a symmetric formulation. The gradient of a quantity can be determined using various formulations [Mon05]. We use

$$\nabla A_i = \sum_j \left(A_j - A_i \right) \frac{m_j}{\rho_j} \nabla_i W_{ij} \tag{3.3}$$

as it guarantees a proper first order interpolation, where $\nabla_i W_{ij}$ is the gradient of the kernel function with respect to particle i.

3.4 Overview

We consider the surface to be the most important fluid region, thus, we want to reserve the highest simulation resolution to the surface and coarser resolutions to the fluid bulk, i.e.,, according to the distance to the surface. Therefore, we first calculate the surface distance ϕ_i for each particle i (see Sec. 3.5.1) and map it to the optimal mass m_i^{opt} the particle should have using a sizing function (see Sec. 3.5.2). We calculate the optimal mass as a continuous quantity without restrictions to levels and try to adapt particles to be as close to m_i^{opt} as possible. Depending on the ratio of a particle's current and optimal mass, we classify particles into five categories

- o: particle is close to optimal size
- s or l: particle is slightly too small or large, respectively
- S or L: particle is strongly too small or large, respectively

Particles of class L are strongly too large and thus are split into smaller particles. As introducing new particles into the simulation often causes instabilities, we use statically optimized $1{:}n$-patterns to reduce the initial error (see Sec. 3.6.1). Particles of class S are strongly too small and their mass is distributed among neighbors before they get completely removed from the simulation, yielding an $(n+1){:}n$-merging of particles (see Sec. 3.6.2). The size of particles of classes s and l is increased or decreased by exchanging mass between neighboring particles in order to meet m_i^{opt} (see Sec. 3.6.3). In summary, the following operations are performed according to the particle class

L Split (Sec. 3.6.1)

S $\begin{cases}\text{Redistribute mass and remove particle (Sec. 3.6.2)}\\ \text{Receive mass from redistribution of } S \text{ (Sec. 3.6.2)}\end{cases}$

l Redistribute mass (Sec. 3.6.3)

s Receive mass from redistribution of l or S (Secs. 3.6.2, 3.6.3)

o Leave unchanged

To reduce errors introduced by splitting or merging particles, we propose the concept of implicit temporal blending that approximately tracks the motion of the original particles, as if they still existed, and only slowly blends to the new particle set (see Sec. 3.7). As changing a particle's mass alters its support radius, we derive a stable SPH formulation to handle variable support radii (see Sec. 3.8) in a symmetric SPH formulation. Algorithm 3.1 shows how our adaptive method can be incorporated into an existing SPH framework.

3.5 Particle classification

As our interest mainly lies on the surface detail, we want to simulate the surface with the finest particle resolution. Therefore, for each particle i we first determine the signed distance to the fluid surface (Sec. 3.5.1). The surface distance is then mapped to a desired optimal particle size m_i^{opt} and classified according to the ratio of its current mass to the optimal mass (Sec. 3.5.2). The mapping yields continuous optimal masses determined by the minimum m^{fine} and maximum m^{base} particle masses which are user-defined parameters to control adaptivity.

3.5.1 Surface detection

To determine the surface of the fluid, we use the Level-set function proposed by Zhu and Bridson [ZB05] which gives an initial estimate of the distance to the surface $\tilde{\phi}_i(t)$ for particles i close to the surface, where the values are negative as particles are on the inside of the fluid. We use the propagation method proposed by Horvath and Solenthaler [HS13] to iteratively propagate distance values ϕ_i to *all* particles in the simulation and clamp the resulting values to the largest distance possible ϕ_{max}. We adopt the approach with minor modifications to allow for adaptive particle sizes.

To avoid spurious detection of surface particles using Level-set functions, we mark particles with more than 45 neighbors as interior particles. Additionally, we do not limit the change of the surface distance for particles in the iterative step of the algorithm as this provides quicker reaction to changes. As we later derive the desired particle mass from the surface distance, particles need to have reasonably smooth distance values. In order to smooth the distance values, we apply an SPH interpolation $\phi_i = \sum_j \frac{m_j}{\rho_j} \phi_j W_{ij}$ in the end.

This algorithm efficiently calculates a stable estimate of the surface distance of all particles as shown in Fig. 3.3.

3.5.2 Sizing functions and particle classification

Using the previously determined surface distance we now calculate the desired particle mass m_i^{opt}. At the furthest distance ϕ_{max} we want to use particles of mass m^{base} and at the surface we want to use particles of mass m^{fine}. Using the adaptivity factor $\alpha = \frac{m^{\text{fine}}}{m^{\text{base}}}$ we determine the optimal particle mass as a linear interpolation of the mass between m^{fine} and m^{base}

$$m_i^{\text{opt}}(\phi_i) = m^{\text{base}} \left(\frac{\min(|\phi_i|, |\phi_{\text{max}}|)}{|\phi_{\text{max}}|} (1 - \alpha) + \alpha \right). \tag{3.4}$$

Particle i is then classified into the categories described in Sec. 3.4 as

$$C_i = \begin{cases} S & m_i^{\text{rel}} < 0.5 \\ s & 0.5 \leq m_i^{\text{rel}} \leq 0.9 \\ o & 0.9 < m_i^{\text{rel}} < 1.1 \\ l & 1.1 \leq m_i^{\text{rel}} \leq 2 \\ L & 2 < m_i^{\text{rel}}, \end{cases} \tag{3.5}$$

where $m_i^{\text{rel}} = m_i / m_i^{\text{opt}}$ denotes the relative mass.

o is chosen with a 10% margin around m_i^{opt} to prevent particles that are close to their optimal mass from causing unnecessary computational effort for redistributing comparably small amounts of mass. Using C_i, our adaptive approach tries to adjust particle masses according to the surface distance. In regions, however, where only l or only s particles are present, m_i can deviate from m_i^{opt} as no mass redistribution is applied (see Fig. 3.3). Note, C_i does not relate to fixed particle sizes as in level-based approaches.

We use the linear scaling of the mass as this creates a very smooth change in resolution over the simulation domain, whereas linearly changing the radius would introduce a cubic change of mass. Although both options yield the same surface detail for unchanged m^{fine}, a cubic change of mass would significantly increase the number of particles. Certain simulations, especially those of very high adaptivity ratios, see Fig. 3.2, could benefit from fine tuned sizing functions that can be interchanged with our linear scaling to improve surface detail even further. Additionally, we could use different inputs to the sizing function, e.g.,the distance to regions of high flow vorticity, as long as the distance values are smooth, which can be achieved by the propagation method of Horvath and Solenthaler [HS13].

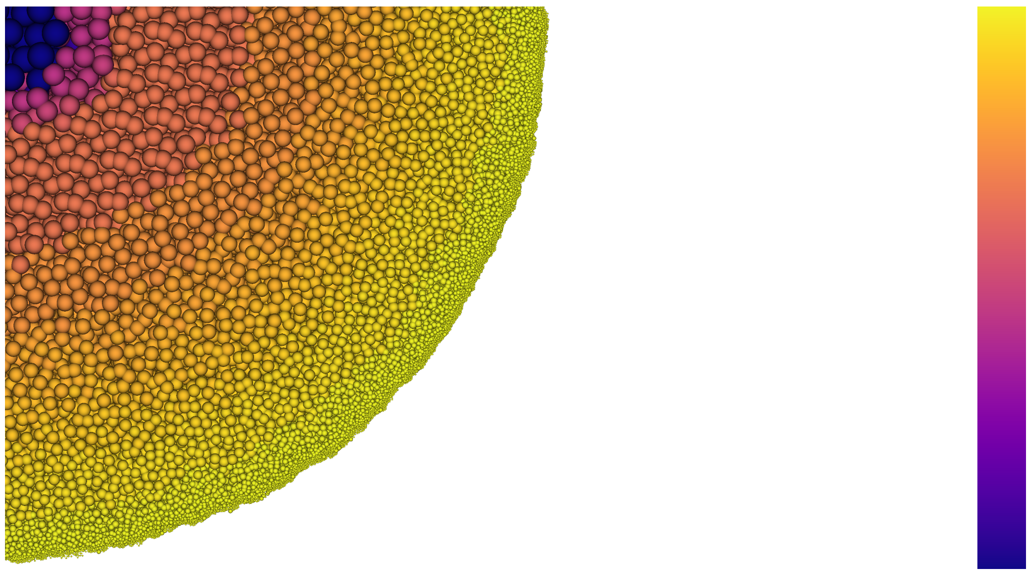


Figure 3.2: This image shows a cross section of the sphere in the sphere drop scenario where we used an adaptivity ratio of 1:10000. Mass is color coded from purple m^{base} to yellow m^{fine}.

3.6 Adaptivity using splitting and merging

In order to adjust particles of classes L and S, we use splitting and merging operations. Particles have usually been split into a fixed number of particles. We, however, want to achieve a smooth adaptive resolution. Therefore, we use preprocessed $1{:}n$-particle refinement patterns from which we can choose the appropriate pattern to create particles close to their optimal mass (Sec. 3.6.1). As particle merging in fixed $n{:}1$-patterns is often impossible due to missing merge partners [Ada+07] or causes large errors that have to be corrected [OK12], we redistribute the mass of *a single* S particle among its n neighbors yielding $(n{+}1){:}n$-merge processes that overcome these difficulties (Sec. 3.6.2). Additionally, to allow for smooth adjustments, particles of class l can redistribute their excess mass among neighbors classified as s (Sec. 3.6.3).

3.6.1 Increasing resolution using particle-splitting

We split a particle i of class L into n child particles that are close to m_i^{opt} by choosing $n = \lceil m_i/m_i^{\text{opt}} \rceil$. To allow for arbitrary $1{:}n$-splitting we generate patterns using an optimization process. For each n we initially generate a pattern in which uniform particles are placed evenly distanced on the surface of a sphere. For patterns with $n > 4$, one of the particles is placed in the center. The particle positions of these initial patterns are then optimized similarly to Vacondio et al. [Vac+16]. For the actual splitting process we use the optimized patterns to generate the positions of the new particles and copy all other quantities of the original L particle which is the only way to conserve kinetic energy and linear and angular momentum [Vac+16]. The mass of the new particles j is directly set as $m_j = m_i/n$ which conserves mass exactly.

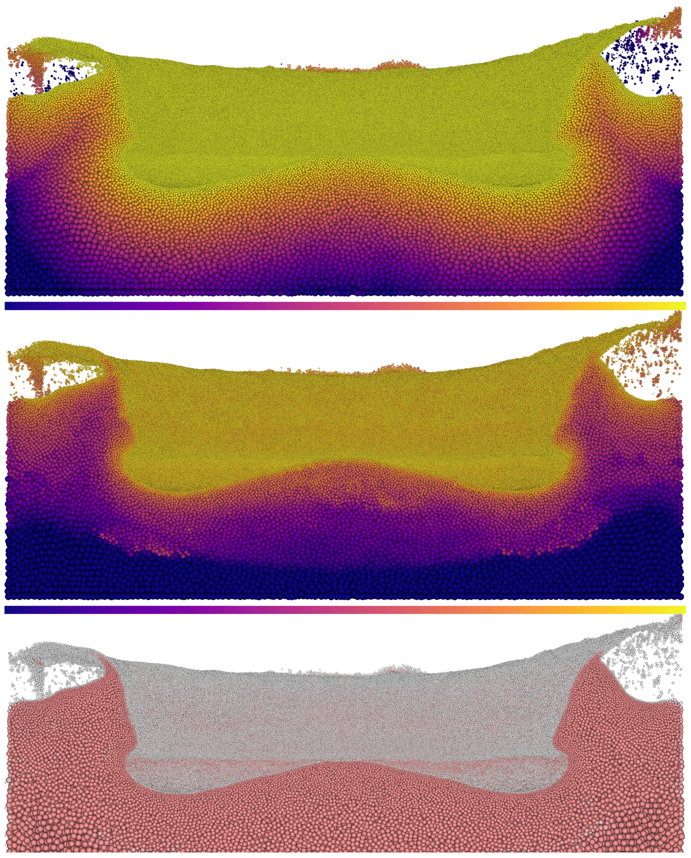


Figure 3.3: Adaptive simulation of a drop of water that drips into a tank with particle mass ratios of 1:300. Surface distance (top) is color coded from 0 (yellow) to $\phi_{\max}$ (purple). Support radius (middle) is color coded from yellow to purple. Density (bottom) is color coded from $0.75\rho_0$(black) to $1.25\rho_0$ (red). Note, our method generates a smooth distance field even with adaptive particles and adapts resolutions according to the distance with respect to the classification. We are able to generate a smooth density over the simulation even when particles of different resolutions interact. The apparent banding in the lowest resolution regions is due to a lack of possible sharing partners as the difference to the ideal mass is too small to cause splitting.

3.6.2 Reducing resolution using particle-merging

Particles i of class S are smaller than $m_i^{\text{opt}}/2$ and are removed from the simulation by redistributing their mass to nearby particles classified as s or S. Although our approach uses continuous particle masses and principally allows merging of arbitrary particle combinations without being restricted to particle levels, we only merge particle i with neighboring s and S particles as they also carry too little mass. We only remove one particle to get an $(n+1)$:n-pattern that smoothly adjusts masses instead of merging in an n:1-pattern.

In order to find neighboring s and S particles of particle i we iterate over all neighboring particles and check if a neighboring particle j is classified as either s or S and if it doesn't already have a distribution partner. We limit the distance to be within $\frac{h_i}{2}$ to reduce long distance merging that tends to be unstable. Additionally, we check if the combined mass $m_j + \frac{m_i}{n} < m^{\text{base}}$ results in a valid size, where n is the number of partners already found plus one. If the particle is a valid partner, we mark it accordingly. If no partner is found, the initial particle i is left in its current state. Due to continuous masses, arbitrary merging combinations and the smooth transition of resolutions, in practice this is not an issue.

An S particle i with n partners distributes $m_n = \frac{m_i}{n}$ to every partner j. Therefore, we iterate over all partners that have previously been marked as merge partners of i and calculate their new masses and other quantities A, i.e.,position, velocity and surface distance, using a mass weighted average as

$$\begin{aligned} m_j^* &= m_j + m_n \\ A_j^* &= \frac{A_i m_n + A_j m_j}{m_j^*}. \end{aligned} \tag{3.6}$$

Once all partners have been updated, the original particle i is removed from the simulation. By redistributing mass, conservation of mass is guaranteed in our method and by using mass weighted average positions and velocities, we also conserve linear and angular momentum [Vac+13].

3.6.3 Mass redistribution

As l particles i have less than $2m_i^{\text{opt}}$ mass, splitting would at least create two particles with masses less than m_i^{opt}. Instead of splitting these particles, we propose to redistribute the excess mass $m_{\text{ex}} = m_i - m_i^{\text{opt}}$ to nearby s particles. This redistribution smoothly adjusts the resolution and prevents l particles from possibly turning into L particles that later would have to be split.

We use a similar mass redistribution process like for merging in Sec. 3.6.2, however, only the excess mass m_{ex} is redistributed to neighboring s particles j as

$$\begin{aligned} m_i^* &= m_i - m_{\text{ex}} \\ m_j^* &= m_j + \frac{m_{\text{ex}}}{n}. \end{aligned} \tag{3.7}$$

As the initial l particle i remains and only part of its mass is redistributed, we only update quantities A_j (positions, velocities and surface distances) of the partner particles j that receive mass as

$$A_j^* = \frac{\frac{m_{\text{ex}}}{n} A_i + m_j A_j}{\frac{m_{\text{ex}}}{n} + m_j} \tag{3.8}$$

in order to conserve momentum. As this process is very similar to the merging in Sec. 3.6.2, both processes can be combined in a single step that finds partners for both l and S particles.

3.7 Implicit temporal blending

Both adding and removing particles through splitting and merging introduces errors even though positions are updated using weighted averaging and optimized refinement patterns are applied. Previously this error has been reduced by temporally blending in new particles [OK12] which however involves the simulation of

ALGORITHM 3.1: Algorithm Overview. New steps required for our method are marked in orange.

```
1  RESORT AND NEIGHBORLIST STEP
2      Sort Particles
3      Build neighborlists for all particles [WHK16]

4  DENSITY COMPUTATION
5      Compute $\rho_i = \sum_j m_j W_{ij}$ for all particles
6      Blend density for blending particles (see Sec. 3.7)

7  PARTICLE INTEGRATION
8      Calculate advection forces (Surface Tension, Viscosity, etc.) $F_i^{\text{adv}}$
9      Blend velocity for blending particles (see Sec. 3.7)
10     Enforce incompressibility using modified IISPH (see Sec. 3.8)
11     Update position of all particles
12
13 ADAPT RESOLUTION
14     Detect Surface (see Sec. 3.5.1)
15     Apply sizing function to classify particles (see Sec. 3.5.2)
16     Create particles using splitting (see Sec. 3.6.1)
17     Find partners for merging and redistribution (see Sec. 3.6.3)
18     Merge particles and redistribute mass
```

both particle resolutions and severely changes the standard SPH interpolation. We thus propose an *implicit temporal blending* for S and L particles which passively advects the original particle set.

Our algorithm stores the position $\boldsymbol{x}_O$ of the original particle O for the new particles, in case of splitting, or for the particles that received mass, in case of merging. These particles use $\boldsymbol{x}_O$ to compute a density ρ_O for the original particle O that ignores all split particles with the same parent. In the first time step ρ_O yields exactly the density as if the old particle still existed and no particles were added. We calculate the density by blending the new particle density ρ_i with the approximate old density ρ_O as

$$\rho_i^{\text{blended}} = (1 - \beta_i)\rho_i + \beta_i \rho_O, \tag{3.9}$$

where β_i denotes the temporal blend weight of particle i. This blended density is used for all calculations the new particles are involved in. Compared to explicit temporal blending no changes to SPH formulations are necessary. We blend the density as all other quantities in an SPH Simulation can be derived from it and improving the quality of the density improves the overall results.

In order to provide a better estimate for the next step, the velocity $\vec{\boldsymbol{v}}_O$ of the original particle is calculated as the mean velocity of all new particles with the same original particle O, or all particles that received mass from the same original particle O, and $\boldsymbol{x}_O$ is updated according to $\vec{\boldsymbol{v}}_O$. We blend the velocities similarly to the density as

$$\vec{\boldsymbol{v}}_i^{\text{blended}} = (1 - \beta_i)\vec{\boldsymbol{v}}_i + \beta_i \vec{\boldsymbol{v}}_O. \tag{3.10}$$

For our blending process we initialize the particle blend weight for particles generated by splitting as $\beta_i = 0.5$ and for particles updated by merging processes as $\beta_i = 0.2$. We chose a smaller initial blend weight for merged particles as they introduce a smaller error. As the tracked position $\boldsymbol{x}_O$ becomes less accurate over time and to fully utilize the new resolution we update the blend weight with a constant rate of $\Delta\beta_i = -0.1$ per timestep until $\beta_i = 0$. While $\beta_i > 0$ the particle does not participate in split, merge or redistribute interactions as it is still in the process of transitioning resolution.

3.8 Adapting to variable support radii

For particles with uniform masses, the support radius is only influenced by the density of a particle (cf. Eq. 3.2) and this influence can typically be ignored [HK89]. Changing support radii due to mass changes in our adaptive simulation, however, cannot simply be ignored when calculating derivatives without causing instabilities. Pressure solvers like IISPH [Ihm+13] and DFSPH [BK15] rely on the time rate of change of density to enforce incompressibility and the accuracy of this derivative is central to their performance.

For our SPH framework with varying radii support, the time rate of change of density can be expressed as

$$\frac{d\rho_i}{dt} = \frac{1}{\Omega_i}\sum_j m_j \vec{\boldsymbol{v}}_{ij}\nabla_i W_{ij}, \tag{3.11}$$

where

$$\Omega_i = \begin{cases} 1 + \frac{h_i}{3\rho_i} m_i \frac{\partial W_{ii}}{\partial h_{ii}} & , C_i^{t-1} = l \\ 1 + \frac{h_i}{3\rho_i} \sum_j m_j \frac{\partial W_{ij}}{\partial h_{ij}} & , \text{else} \end{cases} \tag{3.12}$$

is a corrective factor that depends on the classification C_i^{t-1} of particle i at the last (discrete) time step $t-1$. See App. 3.A.1 for a comprehensive derivation.

This corrective factor is also applied to the pressure force resulting in a corrected pressure force term as

$$F_i^P = \sum_j \frac{m_j}{\rho_j}\left(\frac{P_i}{\rho_i^2\Omega_i} + \frac{P_j}{\rho_j^2\Omega_j}\right)\nabla_i W_{ij}. \tag{3.13}$$

Both Eq. 3.11 and 3.13 are necessary to achieve stable incompressible simulations at resolution interfaces and for varying support radii. For our simulations we modified the IISPH algorithm by carrying the Ω_i term through the derivation resulting in simple adjustments to the overall algorithm. The derivation of the pressure term and the application to IISPH are covered in detail in App. 3.A.2 and 3.A.3 .

3.9 Results and discussion

To compare our adaptive simulation approach with simulations of fixed resolutions, and to show the capabilities of our method, we set up different scenarios. We used a *sphere drop scenario* where we drip a sphere of liquid into a basin (see

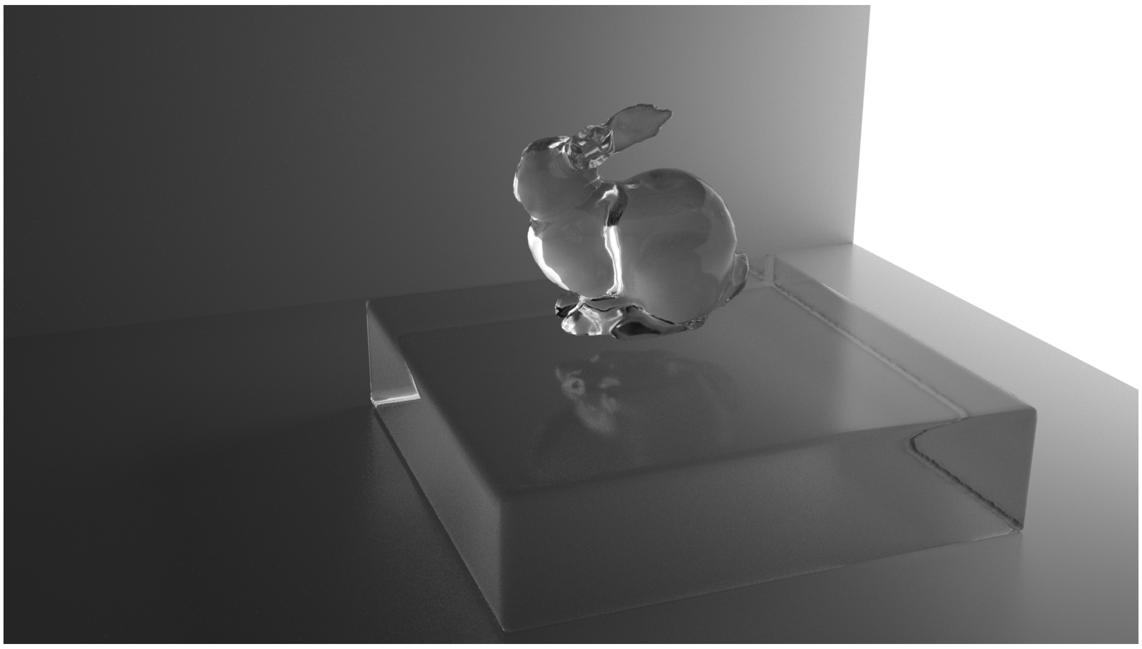


Figure 3.4: This image shows the bunny drop scenario rendered using a ray tracer at an adaptivity ratio of 1:250. The image is at the initial start of the simulation before the splash. The splashing behavior can be seen in Fig. 3.1.

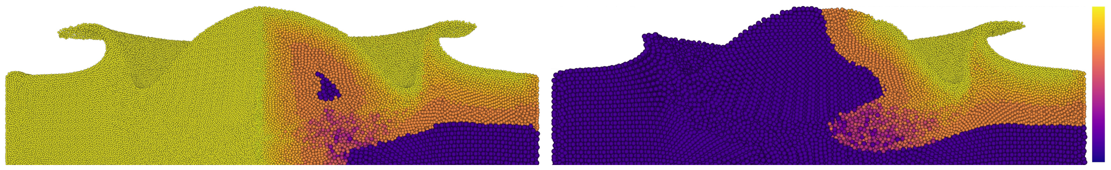


Figure 3.5: This image shows our qualitative comparison. For the left image we changed the sizing function to ensure all particles on the left to be of the highest resolution $m_i = m^{\text{base}}/32$, and for the right image we changed the sizing function to ensure all particles on the left to be of the lowest resolution $m_i = m^{\text{base}}$. Volumes are color coded from purple m^{base} to yellow $m^{\text{base}}/32$.

Fig. 3.7) to compare performance and visual quality, a *bunny drop scenario* where instead of a sphere we used a bunny shaped object (see Fig. 3.1), a *dam break scenario* as a common case, and a *double dam break scenario* (see Fig. 3.9) to show higher adaptive ratios. We assess the quality of our approach by visually comparing adaptive and non-adaptive simulation results. The performance is assessed by comparing run times of simulations with similar resolution.

All simulations were run on a single Nvidia Titan X GPU with 12 GiB of VRAM, an Intel i7-5930k and 16 GiB of RAM. We use the surface tension of Akinci et al. [AAT13], the artificial viscosity of Monaghan [Mon05], boundaries are represented as signed distance fields [HKK07]. Fluid renderings were achieved using Houdini and mantra with no modifications required to surface extraction or ray-tracing techniques.

Qualitative comparison To compare the quality of the results we used the *drop scenario*, see Fig. 3.7. This scenario provided challenging dynamics on the initial impact of the fluid and difficulties of handling the boundary interactions for higher

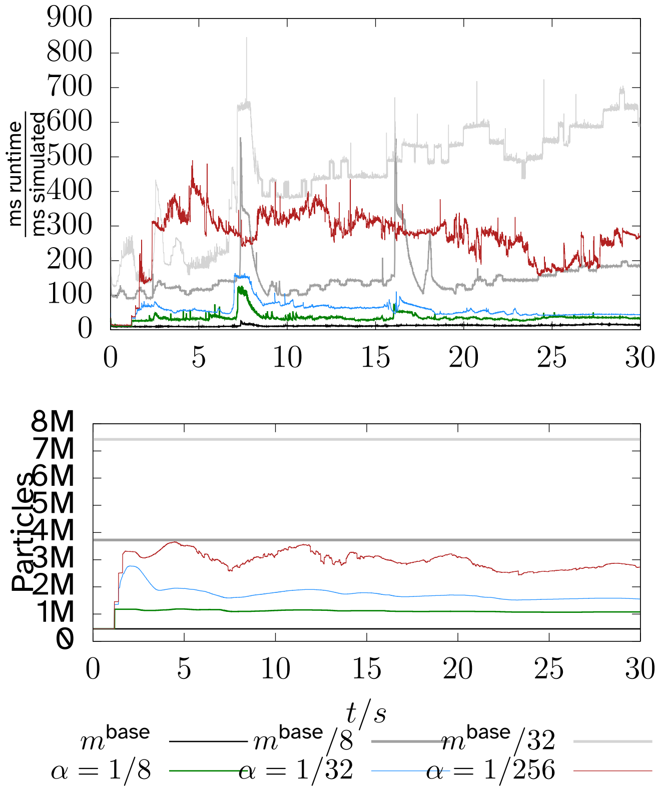


Figure 3.6: These graphs show the performance of the sphere drop scenario with and without adaptivity. The top graph shows the computation time in ms required per ms simulation time. The bottom graph shows the number of particles over the course of the scenario for all the different cases. We used 3 fixed resolutions m^{base}, $m^{\text{base}}/8$ and $m^{\text{base}}/32$ and four adaptive simulations where m^{base} was the same as for the non adaptive tests.

resolutions. Additionally, the quality of the crowning is a simple way of judging the quality of the result and indicates possible negative impacts. Fig. 3.5 shows the comparison setup we used. When comparing the high resolution fixed result with the adaptive result, we see little difference in the crowning, whereas the result is significantly better in comparison to the low resolution result. Overall the behavior was very similar on comparable surface resolutions but differed slightly due to the dependence of certain parameters, e.g. viscosity, on particle resolution. Overall, the fluid behavior was very similar.

Quantitative comparison Figure 3.6 shows the performance and particle counts for the sphere drop scene over $30s$ simulated time using different adaptivity ratios and fixed resolutions. Comparing the results for $\alpha = 1/256$ and $m = m^{\text{base}}/8$ we see a comparable number of particles over the course of the simulation. Although our adaptive simulation was slower than the fixed resolution by a factor of 0.7x, we achieved far more detailed surface dynamics due to the higher surface resolution. When comparing the results for $\alpha = 1/32$ and $m = m^{\text{base}}/32$ which have the same surface resolution, we see a reduction in the average number of particles from $7.4M$ to $1.9M$ particles and a total speedup of 7.3x.

Scaling Considering scaling we saw the largest performance gains for large resolution differences. Due to our sizing function we can roughly estimate the number of particles for an adaptive compact fluid volume as $n_\alpha = n_{\text{base}}\left(1+\log_2\frac{1}{\alpha}\right)$, where n_{base} is the number of equivalent uniform particles of mass m^{base}. With an adaptivity ratio of $\alpha = 1/1024$ the average memory and computational requirements increased by 11x compared to $m = m^{\text{base}}$, like our estimate predicted. Simulating $m = m^{\text{base}}/1024$, i.e.,using the finest resolution, would have required approximately 90x more memory than our adaptive approach which could not not fit into memory. However the average does not take into account the large amount of surface particles that can be generated as spray on thin fluid details and as such the momentary particle counts could be significantly different.

Stability For simulations with highly dynamic behavior like a double dam break (see Fig. 3.8) adaptivity factors of $\alpha = 1/1024$ could stably be used.

Even for adaptivity factors of $\alpha = 1/100000$ we found stable behavior considering the algorithm, however, the computational cost increased significantly past roughly $\alpha = 1/2048$ due to the uniform cell grid we currently use to search for neighbors. Additionally resolution dependent parameters, e.g. viscosity, made choosing the right parameters difficult. Disregarding the problems due to the data structure we found no limit to our adaptivity.

Limitations Although similar, there always were differences between adaptive and equivalent fixed high resolution simulations which mostly depended on viscosity and surface tension parameters. While we could tune parameters to make the fluids behave more similar, different fluid behavior is a common problem if different time steps or particle sizes are used [Pee+15]. When using highly adaptive simulations, the use of rigid particles [Aki+12] becomes difficult if the rigid is only sampled at one resolution. Additionally uniform cell grid structures become restrictive at higher adaptivity ratios.

3.10 Conclusions

We have presented a novel method for adaptive incompressible SPH simulations. Our method uses continuous particle masses that are smoothly and mass-conservingly adjusted which allows for infinitely adaptive simulations. We use the surface distance to calculate the optimal mass of each particle that we try to achieve. To adjust particle masses, we allow for splitting of particles, merging and redistribution of mass among neighbors yielding a fine-grained control of particle mass. Our novel merging scheme, in which a particle's mass is redistributed among its neighbors, alleviates previous problems of finding merge partners and the smooth redistribution of mass and the proposed implicit temporal blending are able to reduce errors in quantity fields. With respect to changing support radii a simple update to certain equations significantly improved the stability. We are able to stably simulate highly dynamic fluids with particle mass ratios of 5 orders of magnitude between the finest and coarsest resolution, yielding large speedups and reductions in memory compared to similar uniform simulations.

3.A Corrective terms

3.A.1 Time rate of change of density

In order to calculate the time rate of change of density we differentiate the standard SPH estimate for density $\rho_i = \sum_j m_j W_{ij}$ with respect to time as

$$\frac{d\rho_i}{dt} = \sum_j m_j \frac{dW_{ij}}{dt} + \underbrace{\sum_j \frac{dm_j}{dt} W_{ij}}_{\lambda_i}, \tag{3.14}$$

where λ_i is the term introduced by the time rate of change of mass of a particle which for adaptive methods is generally non-zero as merging, splitting, and mass redistribution change the mass of individual particles although the total mass stays constant due to mass-conservation. Using the total time derivative of the kernel

$$\frac{dW_{ij}}{dt} = \frac{\partial W_{ij}}{\partial x_{ij}} \frac{dx_{ij}}{dt} + \frac{\partial W_{ij}}{\partial h_{ij}} \frac{dh_{ij}}{dt}, \tag{3.15}$$

we re-factor Eq. 3.14 into terms containing the time rate of change of mass, terms containing the time rate of change of support radius and the remaining terms as

$$\frac{d\rho_i}{dt} = \sum_j m_j \frac{\partial W_{ij}}{\partial x_{ij}} \frac{dx_{ij}}{dt} + \lambda_i + \sum_j m_j \frac{\partial W_{ij}}{\partial h_{ij}} \frac{dh_{ij}}{dt}. \tag{3.16}$$

The kernel derivative with respect to the distance x_{ij} can be calculated as

$$\frac{\partial W_{ij}}{\partial x_{ij}} \frac{dx_{ij}}{dt} = \vec{\boldsymbol{v}}_{ij} \nabla_i W_{ij}, \tag{3.17}$$

where $\vec{\boldsymbol{v}}_{ij} = \vec{\boldsymbol{v}}_i - \vec{\boldsymbol{v}}_j$ is the velocity difference of the interacting particles and $\nabla_i W_{ij} = \frac{\partial W_{ij}}{\partial \boldsymbol{x}_i}$ denotes the kernel derivative with respect to particle i [Mon05]. Due to the symmetric SPH formulation, the kernel derivative with respect to the support h_{ij} can be calculated as

$$\frac{\partial W_{ij}}{\partial h_{ij}} \frac{dh_{ij}}{dt} = \frac{1}{2} \left(\frac{dh_i}{dt} + \frac{dh_j}{dt} \right) \frac{\partial W_{ij}}{\partial h_{ij}}. \tag{3.18}$$

This results in an updated Eq. 3.16 as

$$\frac{d\rho_i}{dt} = \sum_j m_j \vec{\boldsymbol{v}}_{ij} \nabla_i W_{ij} + \lambda_i + \frac{1}{2} \sum_j m_j \left(\frac{dh_i}{dt} + \frac{dh_j}{dt} \right) \frac{\partial W_{ij}}{\partial h_{ij}}. \tag{3.19}$$

Here $\sum_j m_j \vec{\boldsymbol{v}}_{ij} \nabla_i W_{ij}$ is the standard formulation of the time rate of change of density for constant support [Mon05]. The λ_i term is due to the inclusion of time-varying mass and the last term is due to the varying support radii. Considering the third term we calculate the support as a function of density, see Eq. 3.2, and thus

$$\frac{dh_i}{dt} = \frac{\partial h(\rho_i)}{\rho_i} \frac{d\rho_i}{dt} = -\frac{h_i}{3\rho_i} \frac{d\rho_i}{dt}. \tag{3.20}$$

Using this equation we could evolve the last term of Eq. 3.19. However, if we directly use this term, the rate of change of a particle i would depend on the rates of change of neighboring particles. Resolving such a dependency would require an iterative process until a stable result is reached, which is not desirable due to its high computational effort.

In order to resolve this issue, we make the assumption that, due to our mass redistribution and smooth resolution changes, neighboring particles receive similar amounts of mass and change similarly which is the case for particles that receive mass. However for particles that distribute mass (case l) the change of the neighbors is of opposite sign. Let C_i^t denote the classification of particle i at the (discrete) time t, we thus assume

$$\frac{dh_j}{dt} \approx \begin{cases} -\frac{dh_i}{dt} & , C_i^{t-1} = l \\ \frac{dh_i}{dt} & \text{, else.} \end{cases} \tag{3.21}$$

Considering these two cases, we first cover the result for the else case. Assuming $\frac{dh_i}{dt} \approx \frac{dh_j}{dt}$ yields

$$\frac{1}{2}\sum_j m_j \left(\frac{dh_i}{dt} + \frac{dh_i}{dt}\right)\frac{\partial W_{ij}}{\partial h_{ij}} = \frac{dh_i}{dt}\sum_j m_j \frac{\partial W_{ij}}{\partial h_{ij}} \tag{3.22}$$

for the third term of Eq. 3.19 that includes the time rate of change of support radii. Using Eq. 3.20, Eq. 3.19 results in

$$\frac{d\rho_i}{dt} = \sum_j m_j \vec{\boldsymbol{v}}_{ij}\nabla_i W_{ij} + \lambda_i + \left(-\frac{h_i}{3\rho_i}\sum_j m_j \frac{\partial W_{ij}}{\partial h_{ij}}\right)\frac{d\rho_i}{dt}. \tag{3.23}$$

Re-factoring all terms containing $\frac{d\rho_i}{dt}$ to the left hand side yields

$$\frac{d\rho_i}{dt}\underbrace{\left(1 + \frac{h_i}{3\rho_i}\sum_j m_j \frac{\partial W_{ij}}{\partial h_{ij}}\right)}_{\Omega_i} = \sum_j m_j \vec{\boldsymbol{v}}_{ij}\nabla_i W_{ij} + \lambda_i \tag{3.24}$$

and moving the corrective factor Ω_i over we get the final equation for the else case

$$\frac{d\rho_i}{dt} = \frac{1}{\Omega_i}\left(\sum_j m_j \vec{\boldsymbol{v}}_{ij}\nabla_i W_{ij} + \lambda_i\right). \tag{3.25}$$

For the case $C_i^{t-1} = l$ we refer back to Eq. 3.19. By splitting the third term into terms that contain i and terms that don't, we get

$$\frac{1}{2}\sum_{j\neq i} m_j \left(\frac{dh_i}{dt} + \frac{dh_j}{dt}\right)\frac{\partial W_{ij}}{\partial h_{ij}} + \frac{1}{2}m_i\left(\frac{dh_i}{dt} + \frac{dh_i}{dt}\right)\frac{\partial W_{ii}}{\partial h_{ii}}. \tag{3.26}$$

Due to our assumption that $\frac{dh_j}{dt} \approx -\frac{dh_i}{dt}$ for $j \neq i$, the sum-term becomes zero and using Eq. 3.20 we get the full equation for $\frac{d\rho_i}{dt}$ as

$$\frac{d\rho_i}{dt} = \sum_j m_j \vec{\boldsymbol{v}}_{ij}\nabla_i W_{ij} + \lambda_i + \left(-\frac{h_i}{3\rho_i}m_i\frac{\partial W_{ii}}{\partial h_{ii}}\right)\frac{d\rho_i}{dt}, \tag{3.27}$$

which can be re-factored to the same form of Eq. 3.25 with a different corrective factor $\Omega_i = 1 + \frac{h_i}{3\rho_i} m_i \frac{\partial W_{ii}}{\partial h_{ii}}$.

So far, we carried λ_i through all equations to provide the full derivation. However, we found that including this term did not provide any measurable improvement to the stability or behavior of our SPH framework, thus, in practice we drop the term.

3.A.2 Corrected pressure forces

In order to derive the pressure forces we start with the Lagrangian for the non-dissipative motion of a fluid in a potential $\Phi(\boldsymbol{x})$ per unit mass in SPH form [Mon05] as

$$\mathcal{L} = \sum_j m_j \left(\frac{1}{2} \vec{\boldsymbol{v}}_j \cdot \vec{\boldsymbol{v}}_j - u_j - \Phi_j \right), \tag{3.28}$$

where $u_j = u(\rho_j, s_j)$ is the thermal energy per unit mass and s_j is the entropy. Here, the Euler-Lagrange equation $\frac{d}{dt}(\frac{d\mathcal{L}}{d\vec{\boldsymbol{v}}_i}) - \frac{d\mathcal{L}}{d\boldsymbol{x}_i} = 0$ is used to derive the equations of motion. The first term of the Euler-Lagrange equation results in $\frac{d\mathcal{L}}{d\vec{\boldsymbol{v}}_i} = m_i \vec{\boldsymbol{v}}_i$. We assume that the entropy of each element of fluid remains constant, though each particle can have a different entropy [Mon05]. From the first law of thermodynamics it follows that $\frac{\partial u_j}{\partial \rho_j} = \frac{P_j}{\rho_j^2}$ which yields

$$\frac{dm_i}{dt} \vec{\boldsymbol{v}}_i + m_i \frac{d\vec{\boldsymbol{v}}_i}{dt} = -\sum_j m_j \left(\frac{P_j}{\rho_j^2} \frac{\partial \rho_j}{\partial \boldsymbol{x}_i} - \frac{\partial \Phi_j}{\partial \boldsymbol{x}_i} \right). \tag{3.29}$$

Similar to the time rate of change of density, the term containing $\frac{dm_i}{dt}$ also showed no appreciable difference if it was included, thus, we drop it. From the standard SPH estimate for density we can calculate $\frac{\partial \rho_j}{\partial \boldsymbol{x}_i}$ by applying $\frac{\partial}{\partial \boldsymbol{x}_i}$, which similar to Monaghan [Mon05] results in

$$\frac{\partial \rho_j}{\partial \boldsymbol{x}_i} = \sum_k m_k \left(\nabla_j W_{ji} \left[\delta_{ji} - \delta_{jk} \right] + \frac{\partial W_{jk}}{\partial h_{jk}} \frac{dh_{jk}}{d\boldsymbol{x}_i} \right), \tag{3.30}$$

where δ_{ij} is the Kronecker delta. The second term can be developed identically to the similar term in the time rate of change of density resulting in

$$\frac{d\rho_j}{d\boldsymbol{x}_i} = \frac{1}{\Omega_j} \sum_k m_k \nabla_j W_{ji} \left[\delta_{ji} - \delta_{jk} \right]. \tag{3.31}$$

Using these results in Eq. 3.29 and removing the derivative of the potential similarly to Monaghan [Mon92] yields a corrected formulation of the pressure force as

$$\vec{\boldsymbol{F}}_i^P = -m_i \sum_j m_j \left(\frac{P_i}{\Omega_i \rho_i^2} + \frac{P_j}{\Omega_j \rho_j^2} \right) \nabla_i W_{ij}. \tag{3.32}$$

3.A.3 IISPH modifications

Adapting the IISPH method requires only a slight modification. In the original paper by Ihmsen et al. [Ihm+13], Eq. 9 in Sec. 3.1 calculates

$$\Delta t^2 \frac{\vec{\boldsymbol{F}}_i^P}{m_i} = \underbrace{\left(-\Delta t^2 \sum_j \frac{m_j}{\rho_i^2} \nabla_i W_{ij}\right)}_{d_{ii}} P_i + \sum_j \underbrace{-\Delta t^2 \frac{m_j}{\rho_j^2} \nabla_i W_{ij}}_{d_{ij}} P_j. \tag{3.33}$$

We modify the equation to

$$\begin{aligned} \Delta t^2 \frac{\vec{\boldsymbol{F}}_i^P}{m_i} = -\Delta t^2 \sum_j m_j \left(\frac{P_i}{\Omega_i \rho_i^2} + \frac{P_j}{\Omega_j \rho_j^2}\right) \nabla_i W_{ij} = \\ \underbrace{\left(-\Delta t^2 \sum_j \frac{m_j}{\Omega_i \rho_i^2} \nabla_i W_{ij}\right)}_{d_{ii}} P_i + \sum_j \underbrace{-\Delta t^2 \frac{m_j}{\Omega_j \rho_j^2} \nabla_i W_{ij}}_{d_{ij}} P_j \end{aligned} \tag{3.34}$$

by including our corrective factors Ω. There are no further changes required as the pressure force is only computed using the terms d_{ii} and d_{ij} that include the corrective factors.

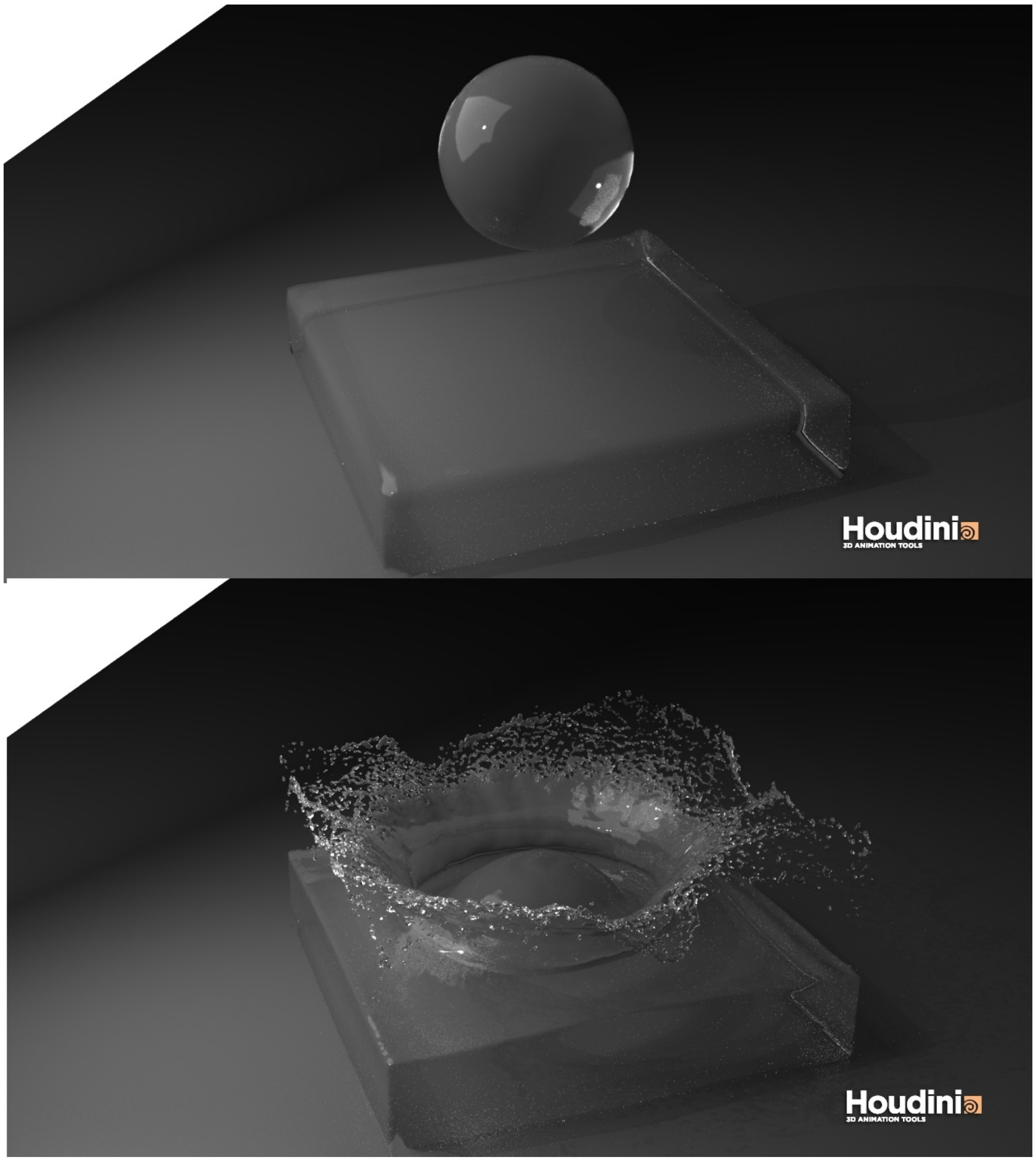


Figure 3.7: This image shows the sphere drop scenario rendered using a ray tracer at an adaptivity ratio of 1:250. We used this scenario in various configurations for performance and quality assessment. The top image is the initial configuration and the bottom image the resulting initial splash.

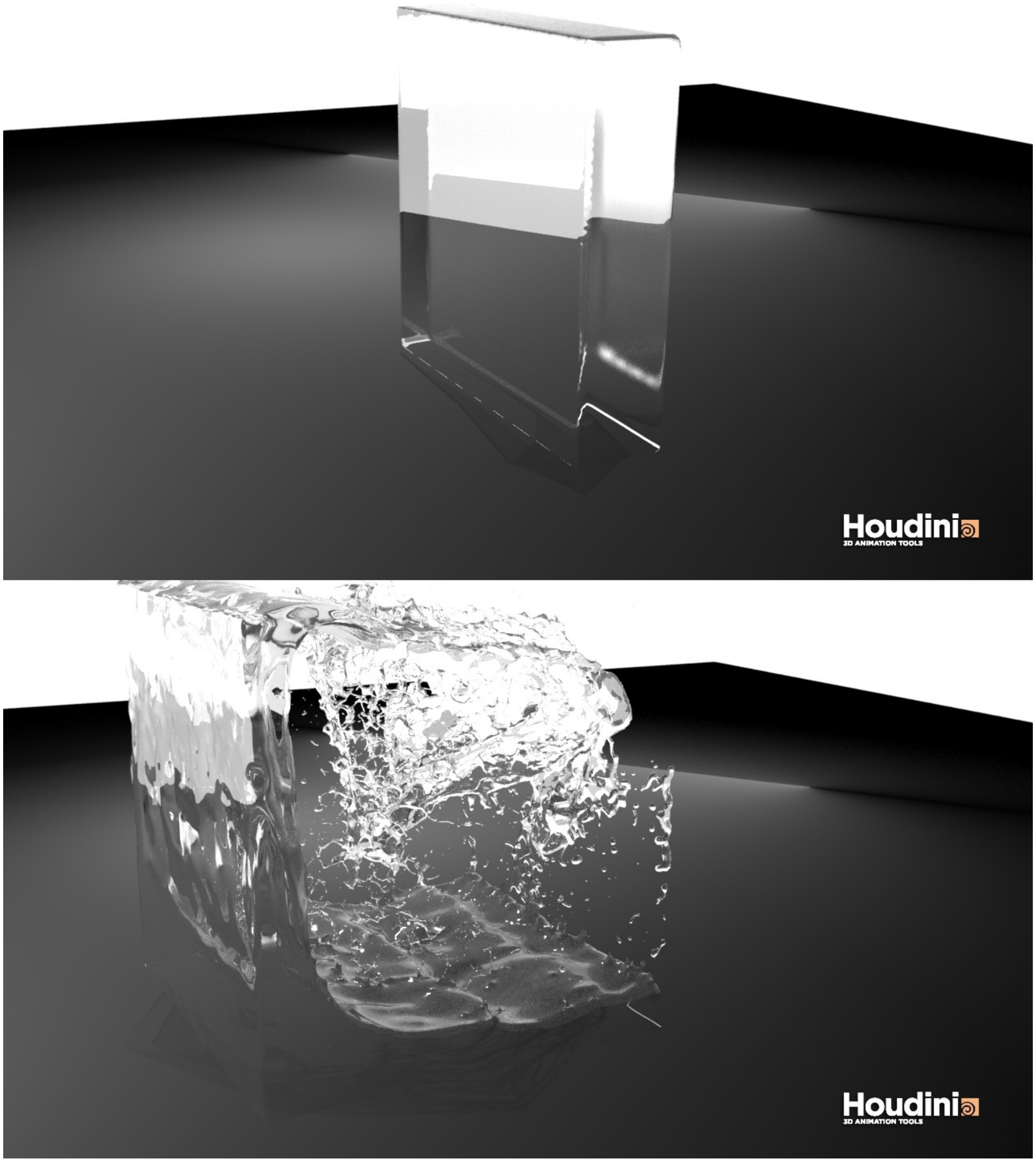


Figure 3.8: This image is a traditional dam break scenario rendered using a ray tracer at an adaptivity ratio of 1:250. The top image shows the initial volume used in the scenario and the bottom image shows the splashing behavior.

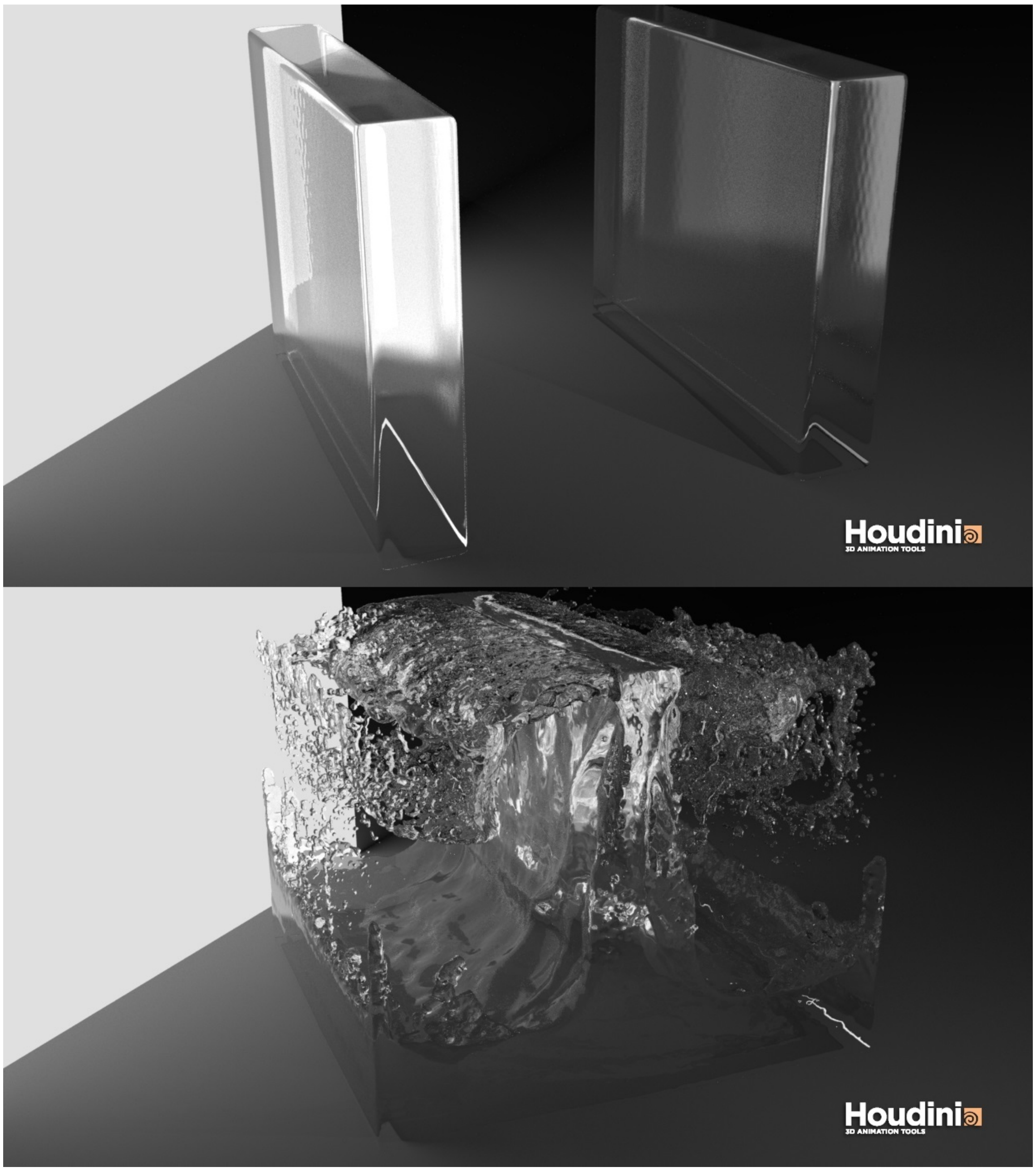


Figure 3.9: This image shows the initial state of our double dam break scenario at the top and the behavior after the collision of the two fluid volumes. An adaptive ratio of $\alpha = 1/500$ and we used a ray tracer to render the images.

Chapter 4

Multi-Level-Memory structures for adaptive SPH simulations

Contextualization

This chapter reprints the paper "Multi-Level-Memory structures for adaptive SPH simulations" published as a conference proceeding of the 2019 Vision Modeling and Visualization conference (VMV 2019) published via Eurographics [WK19]. This paper was honorably mentioned at the conference and, consequently, a significantly extended revision was invited for publication in the Computer Graphics Forum, see Chapter 5. This paper builds on the prior neighbor list approach, see Chapter 2, and describes data handling processes optimized for spatially adaptive SPH simulations using compact hashing, as well as different neighborlist approaches optimized for memory requirements. The main purpose of these approaches was addressing limitations to the practicality of highly adaptive simulations as prior data handling approaches, especially on GPUs, did not yield acceptable computational performance.

Principally, the core idea of the paper is to utilize a self-similar space filling curve, in this case a Morton code, to sort particles with and then efficiently constructing slices through this ordering at different Morton code lengths, based on the resolution of particles, to generate data structures that are well suited for the specific resolution. This is achieved through a combination of a dual layered data structure with a compact cell map that contains particle indices for all occupied cells and a hash map to find cells related to a spatial position, similar to prior work of Teschner et al. [Tes+03]. The approaches proposed here have no impact on the simulation stability, or behavior, making them ideally suited for general usage in a wide variety of context, and especially for spatially adaptive methods.

The main idea of using compact hashing on GPUs, as well as the conceptual ideas of all the neighborlist approaches came from Rene Winchenbach. Andreas Kolb contributed to some of the formal presentations of the methods and co-authored the final paper.

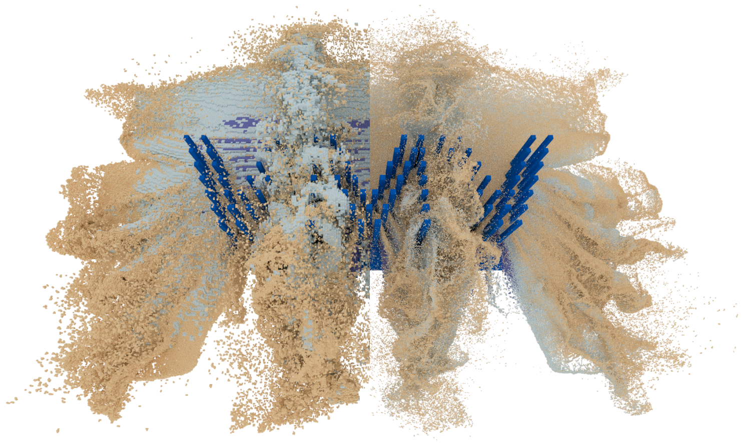


Figure 4.1: Using our multi level memory structure (left image half), we can efficiently simulate a highly adaptive (1000:1) SPH simulation, for up to 8 million particles (right image half), using 4 memory levels. At the pictured timepoint the level 0 domain spans $189 \times 124 \times 84$ cells $(1,968,624)$, whereas the level 3 domain spans 8^3 as many cells. Color coding indicates memory level from 0 (blue) to 3 (red).

Abstract

In this paper we introduce a novel hash map-based sparse data structure for highly adaptive Smoothed Particle Hydrodynamics (SPH) simulations on GPUs. Our multi-level-memory structure is based on stacking multiple independent data structures, which can be created efficiently from the same particle data by utilizing self-similar particle orderings. Furthermore, we propose three neighbor list algorithms that improve performance, or significantly reduce memory requirements, when compared to Verlet-lists for the overall simulation. Overall, our proposed method significantly improves the performance of spatially adaptive methods, allows for the simulation of unbounded domains and reduces memory requirements without interfering with the simulation.

4.1 Introduction

Vivid and highly detailed fluid simulations have become an essential part of modern computer graphics, due to ever increasing demands for more realism. Smoothed Particle Hydrodynamics (SPH) [GM77] is a simulation technique for fluid systems, which has been extended recently to allow for highly adaptive incompressible fluid simulations [WHK17]. Spatially adaptive methods dedicate computational resources where they are most beneficial to the desired outcome. However, adaptive simulations with adaptivity ratios of $1000 : 1$ and higher suffer from significant performance drops due to limitations in the underlying data structures [WHK17]. For CPU based, single resolution SPH simulation methods compact hash maps are commonly used [Ihm+11]. GPU based methods cannot easily utilize these approaches and instead often rely on dense cell structures [Gre10; Gos+10] or linked list based structures [Dom+13; WRR18].

In this paper, we present a hash map based data structure, which is specifically designed to handle the requirements of highly adaptive SPH methods simulated on a GPU. Our proposed data structure works by utilizing a hash map to efficiently access a compact cell list, which refers to particles sorted by a self-similar ordering. We extend this method by efficiently creating multiple distinct data structures, based on different cell sizes, by utilizing the self-similarity. Our method allows us to significantly reduce the number of non-neighbor particle accesses by providing an appropriate data structure for different particle resolutions. Furthermore, we present a corrective algorithm that guarantees symmetric particle neighborhoods, which are essential for spatially adaptive incompressible fluid simulations. Additionally, we propose a set of novel neighbor-list algorithms, which are applicable to adaptive and non-adaptive simulations, by either improving performance or memory consumption. Our proposed method significantly improves the practical applicability of adaptive simulations, and substantially reduces the data structure overhead. Our proposed method provides better memory scaling and allows for the simulation of unbounded domains.

4.2 Related work

SPH has been a very active field of research since its introduction by Gingold and Monaghan [GM77]. Initially, stiff equations of state were employed to achieve simulations of weakly compressible fluids [MCG03; BT07]. Later techniques are based on prediction-correction [SP09], iterative [MM13], and implicit [Ihm+13] methods in order to enforce incompressibility. In addition to solving incompressibility, divergence-free SPH simulations (DFSPH) have been demonstrated [BK15], which significantly stabilize the overall simulation and improve visual fidelity. Recent research has also made large improvements to boundary handling, either by utilizing particles [Ban+18b; Ban+18a; Gis+19], analytical [FM15] or numerical [KB17] boundary models.

Adaptive simulations using splitting and merging processes were introduced by Desbrun and Cani [DC99]. This work was extended by adjusting particle positions after splitting, in order to reduce the error in the pressure term [Ada+07]. To further stabilize the interaction between particles with different resolutions, Keiser et al. [Kei+06] used virtual link particles of neighboring resolutions. To realize adaptivity in incompressible methods, Winchenbach et al. [WHK17] introduce an adaptive method, which works with estimates of original particle positions, a temporal blending process, similar to that proposed by Orthmann and Kolb [OK12], and a process of mass redistribution. This allows for much larger adaptivity ratios than previously possible, in excess of $10,000 : 1$. However, the performance benefits of higher adaptivity ratios are significantly hampered by the limitations of existing data structures.

For CPU based simulations, Ihmsen et al. [Ihm+14] give a good overview of existing data structure methods, and identify a hash map-based method [Ihm+11] as the most efficient data structure. This approach is, however, not directly applicable to GPUs due to the way in which the hash map is constructed. For GPU based simulations, Green [Gre10] introduced a method utilizing a fixed domain with linearly indexed cell lists. A similar approach was used by Dominguez et al. [DCG11], which was optimized for multiple GPUs. Goswami et al. [Gos+10] used Morton codes , however, their approach introduces a complex scheme to balance workloads on the GPU, making it difficult to implement and utilize. In order to limit memory usage on GPUs, Winchenbach et al. [WHK16] introduced an iterative process to constrain the size of so-called Verlet-lists, which are used to store references to neighboring particles. However, all of these methods suffer from scaling and performance problems for adaptive simulations.

Many generic data structures and methods have been developed, for computer animation, where some notable examples include perfect hash maps to store sparse voxel data [LH06; Gar+11], which are not easily scalable to multiple resolutions or approximate nearest neighbors from machine learning aspects [AI08], which are only approximate and designed for high dimensional data. Furthermore, various CPU based approaches exist, e.g.,OpenVDB [Mus+13], but they often require significant changes to be realized on GPU based systems. OpenVDB was realized for GPUs as GVDB, where recently, Wu et al. [Wu+18] introduced a GVDB-based data structure for FLIP-based simulations that significantly improves performance, but which is not directly applicable to SPH, as FLIP imposes different requirements on the data structure, which is an integral part of the simulation itself.

4.3 Basics of Smoothed Particle Hydrodynamics

In general, quantities for a particle i are interpolated from a weighted average using neighboring particles j as [Mon05]

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} W_{ij}, \tag{4.1}$$

where the interpolated quantity is denoted as $A_i = A(\boldsymbol{x}_i)$, which depends on the mass m_j and density ρ_j of neighboring particles within a compact support radius. For further details refer to [Pri12]. The contributions from these neighboring particles are then weighted based on a kernel function $W_{ij} = W(x_{ij}, h_{ij}),\ x_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, which in turn is based on the support radius of an interaction h_{ij}. For adaptive incompressible methods, $h_{ij} = \frac{h_i + h_j}{2}$ is used in order to avoid instabilities. The support radius of a particle can be calculated as

$$h_i = \eta \sqrt[3]{\frac{m_i}{\rho_i}}. \tag{4.2}$$

Here, η is a configuration parameter set based on the chosen kernel function [DA12; WHK16], which determines the number of neighboring particles in a resting state N_h and as such can be found by refactoring of $\frac{4}{3}\pi h^3 = N_h \frac{m_i}{\rho_0}$ [WHK16]. Other kernel functions, e.g.,those of the Wendland family, have much larger N_h values, and therefore require different neighbor list algorithms to prevent excessive memory usage. In computer animation the support radius is often calculated as

$$h_i^0 = \eta \sqrt[3]{V_i^0}, \tag{4.3}$$

which is based on the rest volume of a particle V_i^0 instead. The rest volume of a particle solely relies on the particles physical size, i.e.,$V_i^0 = \frac{4}{3}\pi r^3$ for some radius r, and does not change based on changes in density. As such we denote this support radius as h_i^0. This is equivalent to assuming that $\rho_i = \rho_0$ in (4.2).

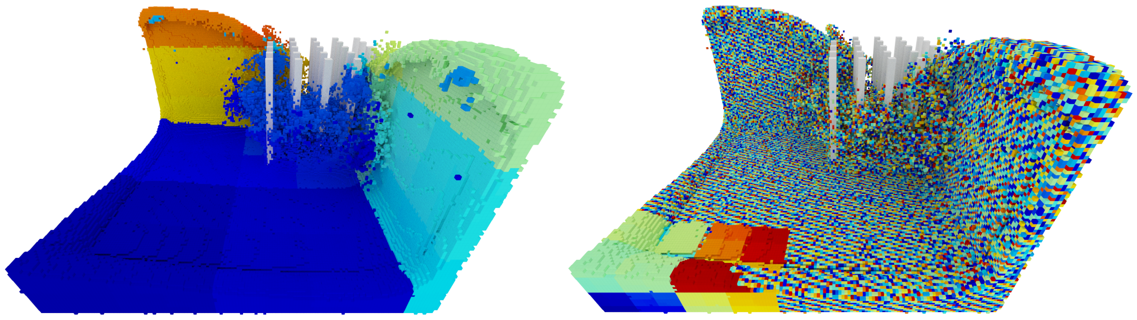


Figure 4.2: These two images show the Morton code $\mathcal{Z}$ on the left and the hashed indices $\mathcal{H}$ on the right for every occupied cell, with color coding indicating indices. The Morton code gives much greater spatial locality but would lead to a significant number of collisions.

4.4 Data structures

The main purpose of a data structure for SPH is to relate the spatial position of a particle with its location in memory in order to reduce the number of particle accesses from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot m)$. One possible approach is to divide the simulation domain into uniform cells of size h [Gre10; Ihm+14]. Note that this notion of a *cell* does not introduce any grid-based methodology into SPH and is solely for data handling. Owing to this, a particle only needs to consider at most 27 cells (a $3 \times 3 \times 3$ sub-grid) for accessing (potentially) neighboring particles. The sphere described by the support radius of a particle will, on average, contain N_h neighbors [DA12] within a volume of $\frac{4}{3}\pi h^3$, whereas the sub-grid of all potential neighbors has a volume of $27h^3$. This means that the sub-grid will contain, on average, $\frac{81}{4\pi}N_h \approx 6.5N_h$ particles, i.e.,15.5% of all potential neighbors are actual neighbors. For an adaptive ratio of $1000 : 1$, however, only 0.016% of all considered particles are neighbors as a cell of the same size would now contain $\frac{81000}{4\pi}N_h$ particles, causing significant performance problems [WHK17]. We first introduce our general data structure for non-adaptive simulations in Sec. 4.4.1, and then the changes required for adaptive simulations in Sec. 4.4.2.

4.4.1 Single-level data structure

We chose the cell size C_{max} to be the same as the largest support radius of any particle as this ensures that all neighbors of all particles are contained within a $3 \times 3 \times 3$ sub-grid, which would not be possible for an arbitrary cell size. We can calculate C_{max} efficiently by using a reduction operation over all particle support radii h_i as

$$C_{\text{max}} = \max\{h_0, ..., h_{n-1}\}. \tag{4.4}$$

The simulation domain itself can similarly be determined as the axis aligned bounding boxes, from $\boldsymbol{D}_{\text{min}}$ to $\boldsymbol{D}_{\text{max}}$, which surrounds the positions of all particles. We can determine these bounds by using reduction operations over all particle positions $\boldsymbol{x}_i$

$$\boldsymbol{D}_{\text{min}} = \min\{\boldsymbol{x}_0, ..., \boldsymbol{x}_{n-1}\}, \boldsymbol{D}_{\text{max}} = \max\{\boldsymbol{x}_0, ..., \boldsymbol{x}_{n-1}\}. \tag{4.5}$$

These bounds are used to calculate the size of the simulation domain in cells as

$$\boldsymbol{D} = \left\lceil \frac{\boldsymbol{D}_{\text{max}} - \boldsymbol{D}_{\text{min}}}{C_{\text{max}}} \right\rceil . \tag{4.6}$$

When using dense data structures, $\boldsymbol{D}$ needs to be kept constant to avoid reallocating memory when particles move outside the current simulation domain. This, in turn, limits the scene's extend as it needs to be known a-priori. We can calculate the integer coordinates $\bar{\boldsymbol{x}}$ for any position $\boldsymbol{x}$ based on the lower simulation bound $\boldsymbol{D}_{\text{min}}$ and the cell size C_{max} as

$$\bar{\boldsymbol{x}} = \left\lfloor \frac{\boldsymbol{x} - \boldsymbol{D}_{\text{min}}}{C_{\text{max}}} \right\rfloor . \tag{4.7}$$

This can be used to determine a linear index $\mathcal{L}$ as

$$\mathcal{L}(\bar{\boldsymbol{x}}) = \bar{\boldsymbol{x}}_x + \boldsymbol{D}_x \left(\bar{\boldsymbol{x}}_y + \boldsymbol{D}_y \left(\bar{\boldsymbol{x}}_z \right) \right) , \tag{4.8}$$

where the subscript denotes the dimension. In a dense cell grid, we can utilize $\mathcal{L}(\bar{\boldsymbol{x}})$ to find the memory location of any position in space. Dense data structures, however, are not desirable as their memory consumption scales with both the simulation domain $\boldsymbol{D}$ and the cell size C_{max}, instead of scaling with the particle count $n_{\text{particles}}$. The Morton code, also sometimes referred to as the Z-ordering, is an alternative indexing scheme, which describes a self-similar space-filling curve. We can efficiently determine $\mathcal{Z}(\bar{\boldsymbol{x}})$ by interleaving the binary representation of an integer coordinates as

$$\bar{\boldsymbol{x}} = \begin{pmatrix} ...\bar{\boldsymbol{x}}_x^3\bar{\boldsymbol{x}}_x^2\bar{\boldsymbol{x}}_x^1\bar{\boldsymbol{x}}_x^0 \\ ...\bar{\boldsymbol{x}}_y^3\bar{\boldsymbol{x}}_y^2\bar{\boldsymbol{x}}_y^1\bar{\boldsymbol{x}}_y^0 \\ ...\bar{\boldsymbol{x}}_z^3\bar{\boldsymbol{x}}_z^2\bar{\boldsymbol{x}}_z^1\bar{\boldsymbol{x}}_z^0 \end{pmatrix} \rightarrow \mathcal{Z}(\bar{\boldsymbol{x}}) = ...\bar{\boldsymbol{x}}_z^3\bar{\boldsymbol{x}}_y^3\bar{\boldsymbol{x}}_x^3\bar{\boldsymbol{x}}_z^2\bar{\boldsymbol{x}}_y^2\bar{\boldsymbol{x}}_x^2\bar{\boldsymbol{x}}_z^1\bar{\boldsymbol{x}}_y^1\bar{\boldsymbol{x}}_x^1\bar{\boldsymbol{x}}_z^0\bar{\boldsymbol{x}}_y^0\bar{\boldsymbol{x}}_x^0,$$

where the superscript denotes a specific bit. Using a 32 bit Morton code results in 10 bit per dimension, meaning each dimension can contain a maximum of $\#K = 1024$ cells. A 64 bit Morton code results in 21 bit per dimension, meaning a maximum of $\#K = 2097152$ cells per dimension. On one hand it would be possible to create an octree directly from Morton codes [Kar12], as this code represents the ordering of an octree. For SPH simulations many nodes of an octree, e.g.,the root node, do not contain any useful information and furthermore, traversing an octree is computationally expensive and the memory consumption of an octree is not independent of the content. On the other hand, a dense data structure using a Morton code would require excessive amounts of memory.

We instead propose to create a list of all occupied cells, as the number of occupied cells n_{occupied} is bound by the number of particles $n_{\text{particles}}$, as the worst case would be every particle occupying a different cell. To generate this list, we first re-sort all particles according to their Morton code $\mathcal{Z}_i = \mathcal{Z}(\bar{\boldsymbol{x}}_i)$. Using this ordering we create a list $\mathbb{C}$ of length $n_{\text{particles}} + 1$, where each element is determined as

$$\mathbb{C}[i] = \begin{cases} i & \text{, if } i = 0 \vee \mathcal{Z}_i \neq \mathcal{Z}_{i-1} \\ -1 & \text{, if } \mathcal{Z}_i = \mathcal{Z}_{i-1} \\ n_{\text{particles}} & \text{, else.} \end{cases} \tag{4.9}$$

$\mathbb{C}$ now contains either a marker entry (-1 or $n_{\text{particles}}$), or the first index of a particle in an occupied cell, which is similar to the approach by Green [Gre10]. We can now compact $\mathbb{C}$, by removing all invalid entries, which gives us a list $\mathbb{C}^{\text{begin}}_{\text{compact}}$ of length $n_{\text{occupied}} + 1$. Using this list of occupied cell beginnings, we can calculate the number of particles in each occupied cell as

$$\mathbb{C}^{\text{length}}_{\text{compact}}[i] = \mathbb{C}^{\text{begin}}_{\text{compact}}[i+1] - \mathbb{C}^{\text{begin}}_{\text{compact}}[i]. \tag{4.10}$$

This compact list, however, does not yield any way to find the memory location for a particle based on its spatial location. To resolve this, we propose to apply a hash map on top of $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and $\mathbb{C}^{\text{length}}_{\text{compact}}$. Following Ihmsen et al. [Ihm+11], we determine the hash of an integer coordinate by using three large prime numbers $p_1 = 73856093$, $p_2 = 19349663$, $p_3 = 83492791$ and the size of the hash table n_{hash} as

$$\mathcal{H}(\bar{\boldsymbol{x}}) = \left(p_1 \bar{\boldsymbol{x}}_x + p_2 \bar{\boldsymbol{x}}_y + p_3 \bar{\boldsymbol{x}}_z\right) \% n_{\text{hash}}, \tag{4.11}$$

where we choose n_{hash} as the smallest prime number larger than the maximum number of particles in a simulation, as this gives a relatively sparse hash map with few collisions, in general.

However, this hash function leads to low coherency for nearby particles, meaning that particles which are spatially close, might be assigned to very distant memory locations, see Fig. 4.2, right. If we can avoid hash collisions, we can embed the cell information directly into the hash map, which removes a level of indirection. However, if we chose a simple spatially coherent hash function, e.g.,$\mathcal{H}(\bar{\boldsymbol{x}}) = \mathcal{Z}(\bar{\boldsymbol{x}}) \% n_{\text{hash}}$, we would find significantly more hash collisions, which outweighs the benefit of more coherence. Contrarily, choosing a perfect hash function, e.g., [LH06], reduces the number of collisions, but at the cost of a significant overhead for its creation. Therefore, we opt for the hash function of Ihmsen et al. [Ihm+11] as it provides a good balance between complexity and collisions. Furthermore, as we utilize per-particle neighbor-lists, we need to access the cells only once during each timestep, as all particle interactions afterwards are calculated using these neighbor-lists avoiding the spatial incoherence of the function.

The hash map itself is similar to the cell list in that it contains a begin entry and a length entry, where the begin entry now points to the first cell mapped to a hash table entry and the length entry indicates how many cells map to this hash table entry. If there is no cell then the length is 0, if there is a single cell occupying this hash map the length is 1 and a length > 1 indicates a hash collision. The hash map, contrary to the cell list, is not compacted and as such allows us access via the hash index of an integer coordinate $\mathcal{H}(\bar{\boldsymbol{x}})$. The process required to find a specific cell c based on the cells integer coordinates $\bar{\boldsymbol{x}}_c$ is described in Algorithm 4.1.

To create the hash table $\mathbb{H}$ we first start by initializing all hash table entries as invalid, i.e.,0 length, and re-sort the list of occupied cells according to the hashed index of the first particle in this cell. We can then, for each occupied cell i, set

$$\mathbb{H}^{\text{begin}}[\mathcal{H}_i] = i, \text{ if } i = 0 \vee \mathcal{H}_i \neq \mathcal{H}_{i-1}, \tag{4.12}$$

where we then set the length entry, for each occupied cell i, as

$$\mathbb{H}^{\text{length}}\left[\mathcal{H}_i\right] = i - \mathbb{H}^{\text{begin}}\left[\mathcal{H}_i\right] - 1, \text{ if } i = n_{\text{occupied}} \vee \mathcal{H}_i \neq \mathcal{H}_{i+1} \tag{4.13}$$

which naturally handles hash collisions as the predicate is based on $\mathcal{H}_i \neq \mathcal{H}_{i+1}$ which is only true for the last cell associated with a certain hash value. The overall algorithm is described in Algorithm 4.2.

ALGORITHM 4.1: The algorithm to access the cell associated with an arbitrary integer coordinate using our proposed sparse data structure without embedding $\mathbb{C}$ into $\mathbb{H}$. Note that an empty cell can map to the same hash map entry as an occupied cell, without causing a collision, and as such we always check $\mathcal{Z}_c = \mathcal{Z}_j$ to avoid returning a wrong cell.

```
1  Calculate $\bar{\boldsymbol{x}}_c$ for the cell we are looking for
2  Calculate $\mathcal{Z}_c$ and $\mathcal{H}_c$ for $\bar{\boldsymbol{x}}_c$
3  Look-up $b = \mathbb{H}^{\text{begin}}[\mathcal{H}_c]$ and $l = \mathbb{H}^{\text{length}}[\mathcal{H}_c]$
4  If $l \neq 0$
5      For $h \in [b, b+l)$
6          Look-up Particle $j = \mathbb{C}^{\text{begin}}_{\text{compact}}[h]$
7          Calculate $\bar{\boldsymbol{x}}_j$ and $\mathcal{Z}_j$
8          If $\mathcal{Z}_c = \mathcal{Z}_j$
9              Return $\mathbb{C}_{\text{compact}}[b]$
10 Return not found
```

4.4.2 Multi-level data structures

The prior section described our approach for uniform cell sizes, which would suffer from the same problems for adaptive simulations as prior methods, due to a mismatch of cell size and particle resolution. However, as we based our method on a Morton code, we can utilize the self-similarity to efficiently create multiple, distinct, data structures for different cell sizes on the same underlying particle data. This is possible for, coarser, power of 2 multiple cell sizes of the cell size used for re-sorting the data.

We start with an initially much finer particle sorting $\mathcal{Z}^{\text{fine}}$, from which we can generate the desired coarser resolutions. To determine $\mathcal{Z}^{\text{fine}}$ we calculate the corresponding cell size C_{fine}, based on the largest dimension $P = \max(\boldsymbol{D}_x, \boldsymbol{D}_y, \boldsymbol{D}_z)$, as

$$C_{\text{fine}} = C_{\text{max}} \frac{2^{\lceil \log_2(P) \rceil}}{\#K}. \tag{4.14}$$

Here, $\#K$ depends on the size of the Morton code used (see Sec. 4.4.1), C_{fine} is the smallest cell size that can be represented using this code length, and $2^{\lceil \log_2(P) \rceil}/\#K = 2^{-L_{\text{fine}}}$, $L_{\text{fine}} \in \mathbb{N}$ is the refinement factor. The algorithm described in the Sec. 4.4.1 can now be extended by creating the cell list and hash map for a Morton code $\mathcal{Z}^{\text{max}}$ based on C_{max} and additional finer levels $0 < L \leq L_{\text{fine}}$ using the integer coordinates

$$\bar{\boldsymbol{x}}^L = \left\lfloor \frac{\boldsymbol{x} - \boldsymbol{x}_{\text{min}}}{C_{\text{max}} \cdot 2^{-L}} \right\rfloor. \tag{4.15}$$

$L = 0$ results in the same data structures as the single-level version of this algorithm and $L = L_{\text{fine}}$ the finest possible data structure, with the given Morton code.

ALGORITHM 4.2: Our proposed single resolution data-structure algorithm. This algorithm first re-sorts all particles and then creates a compact cell table followed by the creation of our hash map as described in Section 4.4.1.

```
1  Initialize
2      Calculate $C_{\text{max}}$, $\mathbf{D}_{\text{min}}$ and $\mathbf{D}_{\text{max}}$ using reductions
3      Calculate $P^2$ based on $\boldsymbol{D}$
4      Re-sort particles using $\mathcal{Z}^{\text{fine}}$
5  Cell table creation
6      Initialize $\mathbb{C} = -1$ and $\mathbb{C}^{\text{length}}_{\text{compact}} = 0$
7      Create $\mathbb{C}$ based on Morton codes of consecutive particles
8      Compact $\mathbb{C}$ into $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and determine $\mathbb{C}^{\text{length}}_{\text{compact}}$
9      Calculate $\mathcal{H}_i$ for all particles
10     Re-sort $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and $\mathbb{C}^{\text{length}}_{\text{compact}}$ based on the hash index of the first contained
         particle.
11 Hash map creation
12     Initialize $\mathbb{H}^{\text{begin}} = -1$ and $\mathbb{H}^{\text{length}} = 0$
13     Create $\mathbb{H}^{\text{begin}}$ based on compacted cell list
14     Calculate $\mathbb{H}^{\text{length}}$
15     Embed $\mathbb{C}_{\text{compact}}$ into $\mathbb{H}$ if $\mathbb{H}^{\text{length}} = 1$
```

The corresponding Morton code is determined as $\mathcal{Z}^L(\boldsymbol{x}) = \mathcal{Z}(\bar{\boldsymbol{x}}^L)$. We relate the maximum level L_{max} to the maximal adaptivity ratio α of the simulation as

$$L_{\text{max}} = \min\left\{\left\lceil log_2 \sqrt[3]{\alpha}\right\rceil, L_{\text{fine}}\right\}, \tag{4.16}$$

and generate the data structure for all levels $0 \leq L \leq L_{\text{max}}$. We store all data structures within single continuous arrays, which allows us to calculate the hashed indices by simply adding an offset based on the level to (4.11) as

$$\mathcal{Z}^L(\bar{\boldsymbol{x}}) = Ln_{\text{hash}} + \left(p_1\bar{\boldsymbol{x}}^L_x + p_2\bar{\boldsymbol{x}}^L_y + p_3\bar{\boldsymbol{x}}^L_z\right)\%n_{\text{hash}}. \tag{4.17}$$

Furthermore, we determine the appropriate level for a particle i according to its support radius as

$$L_i = \text{clamp}\left(\left\lfloor -\log_2 \frac{h_i}{C_{\text{max}}}\right\rfloor, 0, L_{\text{fine}} - 1\right). \tag{4.18}$$

Thus, every particle can easily access all neighbors at any scale $L \in [0, L_{\text{max}}]$ using the data structure for L. However, when a particle i only looks for neighbors at its level L_i, we may encounter asymmetric interactions, as neighbor searches are limited by the cell size for level L_i; see Fig. 4.3. This could be avoided entirely by utilizing a gather-formulation of SPH, but this formulation is not stable for adaptive incompressible SPH [WHK17]. Therefore, we explicitly need to handle this case.

More formally, asymmetric interactions occur when a particle i of lower level L_i is interacting with a particle k of a higher level L_k, i.e.,if $L_k > L_i \wedge x_{ik} < h_{ik}$. In order to resolve the asymmetry, we iterate over all neighboring particles k of i and determine their integer coordinate distance, for the cell size associated with

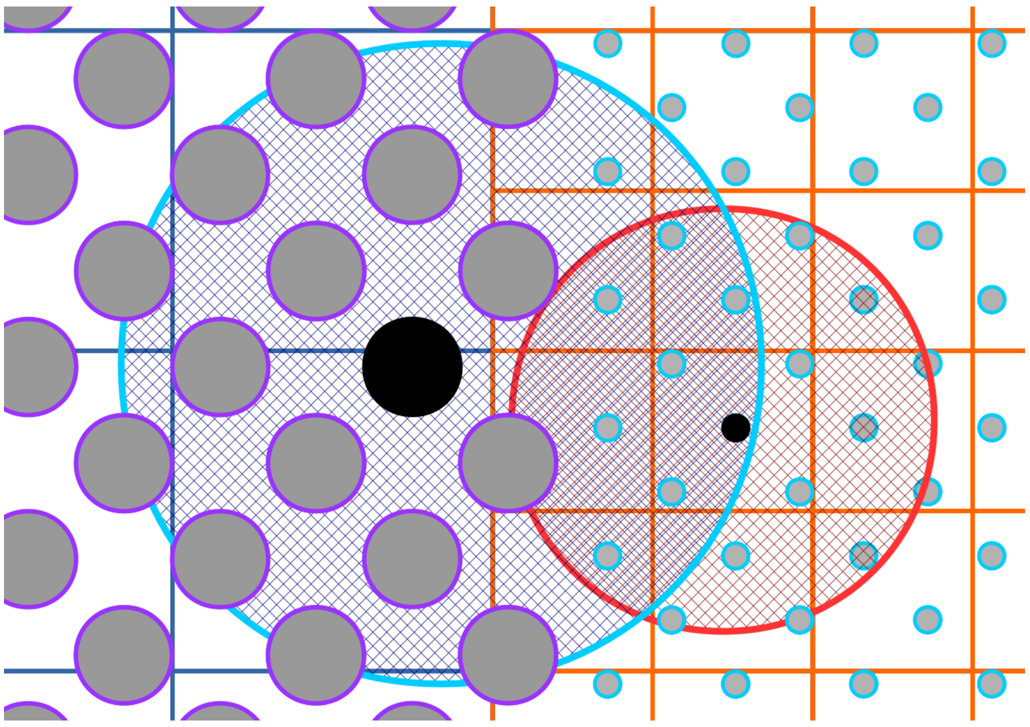


Figure 4.3: Asymmetry interaction: Two particles at different levels may not mutually see each other due to the different cell size. Here, the lower level particle to the left sees the higher level particle to the right, but not vice versa.

L_k as $\bar{\boldsymbol{x}}_{ik}^{L_k} = \bar{\boldsymbol{x}}_i^{L_k} - \bar{\boldsymbol{x}}_k^{L_k}$. We use $\bar{\boldsymbol{x}}_{ik}^{L_k}$ to identify the problem case that k does not see particle i by checking

$$\left\|\bar{\boldsymbol{x}}_{ik}^{L_k}\right\|_1 = \max(|\bar{\boldsymbol{x}}_{ik,x}^{L_k}|, |\bar{\boldsymbol{x}}_{ik,y}^{L_k}|, |\bar{\boldsymbol{x}}_{ik,z}^{L_k}|) > 1. \tag{4.19}$$

We then resolve this issue by atomically updating L_k to be at least L_i, as this ensures that k will search distant enough cells to find i. The overall changes to the single resolution algorithm are relatively minor but are outlined in Algorithm 4.3.

4.5 Neighbor-lists

As noted in Section 4.4, only about 15.5% of all potentially neighboring particles are actual neighbors. To avoid the repeated access to non-neighboring particles, neighbor-lists are a common solution, which store a reduced set of potential neighbors. Verlet-lists store references to every actual neighbor, but for adaptive simulations where the number of neighbors often exceeds $2N_h$ this would lead to excessive memory usage. In order to avoid this, we first introduce a novel histogram based constrained neighbor-list in Section 4.5.1, followed by a span based neighbor-list in Section 4.5.2 and a bitmask based neighbor-list in Section 4.5.3.

4.5.1 Histogram based neighbor-lists

A constrained neighbor list, e.g., [WHK16], limits the number of neighbors N_i for a particle i to an upper bound N_c, where the main motivation comes from reducing

ALGORITHM 4.3: Changes required to create multiple levels of our data structure

```
1 Initialize
2     Calculate $C_{\text{max}}$, $\mathbf{D}_{\text{min}}$ and $\mathbf{D}_{\text{max}}$ using reductions
3     Calculate $P^2$ based on $\boldsymbol{D}$ Re-sort particles using $\mathcal{Z}^{\text{fine}}$
4     Calculate Ideal level $L_i$ for every particle
5 For every Multi-Level-Memory level $L$
6     Execute Cell table creation and Hash map creation using $\mathcal{Z}^L$ and $\mathcal{H}^L$
        respectively.
7 Finalize
8     Enforce symmetric interactions
```

memory requirements and optimizing access patterns and is not based around a change of neighborhood size due to a changing support radius, e.g.,by using (4.2).

Naïvely, it would be possible to simply exclude actual neighbors from this list, however this would lead to asymmetries and thus to instabilities. To avoid this, a constrained neighbor-list method reduces the support radius of a particle h_i until $N_i < N_c$. The constrained neighbor-list approach of Winchenbach et al. [WHK16] implemented this in an iterative process, however, due to the cost of an iteration over all potential neighbors, this method became computationally expensive for adaptive methods. In order to realize this in a single step, we first consider the support radius for i in an interaction with another particle j that would result in $|x_{ij}| = h_{ij}$, and as such $W_{ij} = 0$. We can determine this support radius for each interacting pair of particles as

(4.20) $$k_{ij} = 2|\mathbf{x}_{ij}| - h_j,$$

where the constrained support radius h_i^{c} would trivially be the N_c-th smallest value. However, calculating k_{ij} for all potentially neighboring particles and storing this list to find the N_c-th smallest value is not practical and we instead propose an alternative histogram-based approach. We create a histogram, whose bins evenly segment the range $[0.5h_i, h_i)$ and store the number of particles with

(4.21) $$\frac{1}{2}h_i + \frac{B}{2\#B}h_i \leq k_{ij} < \frac{1}{2}h_i + \frac{B+1}{2\#B}h_i$$

in each bin, where B denotes the bin index and $\#B$ the number of bins for the histogram. This allows us to calculate the bin a particle j belongs to in the histogram for particle i, as

(4.22) $$\#b_{ij} = \left\lfloor \text{clamp}\left(\frac{k_{ij} - 0.5h_i^{\text{new}}}{2\#B \cdot h_i^{\text{new}}}, 0, \#B - 1\right)\right\rfloor.$$

We can store this histogram within the shared memory of a GPU by choosing a small enough bin size and number of bins, e.g.,32 bins with an 8 bit bin size on modern GPUs. Each bin contains the number of particles associated with this range of values and as such, once the histogram is completed, we can sum up the counters, starting with the lowest bin, until the sum is larger than the upper bound of neighbors N_c at some bin index b. The final constrained support radius can then be calculated as

(4.23) $$h_i^c = \frac{h_i}{2}\left[1 + \frac{1}{b-1}\right].$$

In order to avoid an ever decreasing support radius, as constraining can only reduce h_i, we propose to update h_i at the end of every timestep based on the rest support radius (4.3) as

$$h_i(t+\Delta t) = \alpha h_i(t) + (1-\alpha) h_i^0, \tag{4.24}$$

where α is a linear blend weight, usually chosen as 0.95. In general, this neighborlist is still, conceptually, a Verlet-list and requires more than N_h entries as we cannot reduce N_i below N_h without causing instabilities. For larger kernel functions, i.e.,Wendland kernels, which have very large neighborhoods and adaptive simulations this becomes quite memory consuming.

4.5.2 Span based neighbor-lists

Instead of storing explicit references to all actual neighbor particles, we store appropriate index-spans in order to be more memory efficient at the cost of covering more non-interacting neighbor particles. Our general approach is to store one index span for each of the 27 neighboring cells per particle.

For any neighboring cell c we iterate over the contained particles $j \in c$ in ascending order, and store the first index b where $|x_{ib}| < h_{ib}$. We then keep iterating until we find the last index l where $|x_{il}| < h_{il}$, which gives the span of particle indices $j \in [b, l]$ that contains all neighbors of i in c. We store b as well as the length of this span $s = l - b + 1$.

The memory requirement for storing a span is size(b) $= \lceil \log_2 n_{\text{particles}} \rceil$ and size(s) $\propto \lceil \log_2 N_h \frac{3}{4\pi} \rceil$. For non-adaptive simulations size(b) + size(s) is almost always less than 4 byte, however for adaptive simulations the number of particles in a cell can become much larger and we require 8 byte in this case. This is a significant improvement in memory efficiency for non-adaptive and adaptive simulations utilizing kernel functions with $N_h > 27$, as the memory consumption results to $27 \cdot 4$ or $27 \cdot 8$ byte instead of $N_h \cdot 4$ and $2N_h \cdot 4$ byte for the non-adaptive and the adaptive case, respectively.

4.5.3 Bitmask based neighbor-lists

The previously described span based neighbor list requires 8 bytes for adaptive simulations, even though 4 bytes would be sufficient in regions with rather homogeneous support radii. Therefore, we propose a third approach that stores bitmask indicating neighboring particles for cells that have interacting neighbors using 4 byte only, accepting that particles in regions with rather inhomogeneous support radii need additional handling.

Considering the sub-grid of $3 \times 3 \times 3$ cells, we can find a unique mapping from this sub-grid to a linear index $\mathcal{L} \in [0, 27)$, which can be stored in 5 bits, leaving 27 bits for representing the bitmask. In homogeneous regions, a cell contains $N_h \frac{3}{4\pi}$ particles, i.e.,12 for the cubic spline kernel. The 26 bits indicate, if the corresponding particle is an interacting particle. To handle cells with more than 26 neighbors in regions with strongly varying support radii, we have to process all particles in the neighboring cell. We indicate this by setting the full bitmask to 1.

Using this masked neighbor list guarantees a memory consumption of $27 \cdot 4$ byte. The drawback is that cells with more than 26 particles, e.g.,in rather turbulent regions with strongly varying support radii, are handled rather inefficiently.

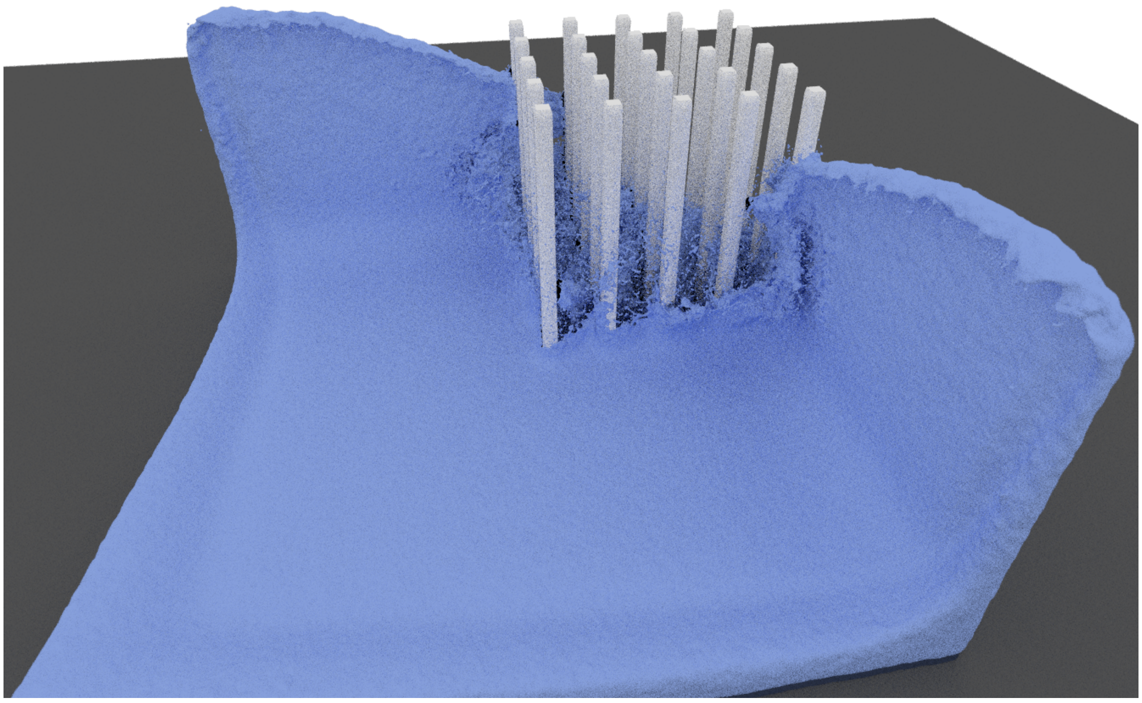


Figure 4.4: A dam break scenario where the fluid was initialized opposite of the rigid obstacles.

However, even in the worst case this is not worse than using no neighbor list at all.

4.6 Results and discussion

All simulations were run using a single Nvidia RTX 2080 Ti GPU with 11 GiB of VRAM, an Intel i7-4790 and 16 GiB of RAM. We used DFSPH [BK15] with a density error limit of 0.01% and a divergence error limit of 0.1%, with rigid objects represented as density maps [KB17]. Artificial viscosity was modeled based on XSPH [Mon05], Surface tension was modeled based on [AAT13], fluid air phase interactions were modeled based on [Gis+17]. For adaptivity we utilize [WHK17]. Rendering was done using a proprietary renderer, with surface extraction based on [YT13]. We used the cubic spline kernel for all tests. The full source code of our simulation, and renderer, can be found `www.cg.informatik.uni-siegen.de/openMaelstrom`.

Test Scenes: We evaluated our approach using two text scenes. The *Pillars* test scene is a dambreak, depicted in Figure 4.4, where an initial fluid volume interacts with many small rigid obstacles on impact. Here, the simulation domain spans 153^3 cells. In this scene we used 2 million particles for the non adaptive tests and up to 8 million particles in the adaptive tests. The *Dragon* test scene is a dambreak scenario, depicted in Figure 4.5, where an initial fluid volume interacts with a single complex rigid object. Here, the simulation domain spans $27 \times 49 \times 55$ cells. In this scene we used 400 thousand particles for the non adaptive tests and up to 8 million particles in the adaptive tests. For numerical stability we set $N_c = 1.2N_h$ for non-adaptive scenes and $N_c = 2.3N_h$ for adaptive scenes. We in-

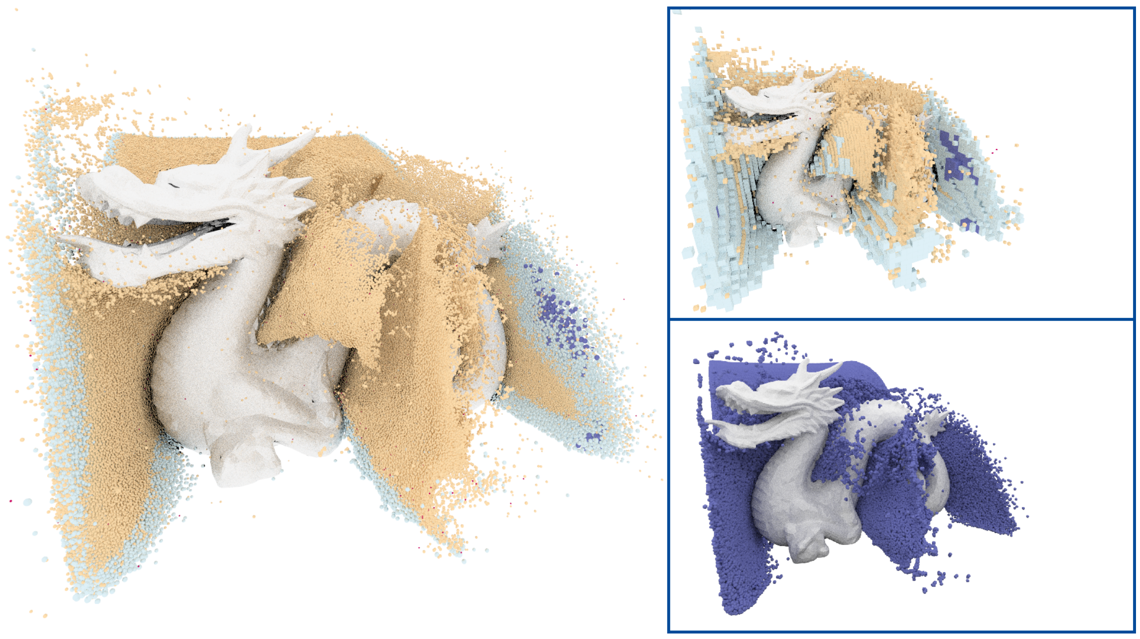


Figure 4.5: This image shows the dragon test scene where a fluid volume collides with a complex rigid object. Color coding indicates the memory levels (blue to red). The top right view visualizes the data structure cells. The bottom right view is a uniform resolution simulation with a comparable time per timestep as the adaptive variant at this timepoint.

tentionally selected the adaptivity ratio to be rather moderate, i.e.,we used 1,000:1, since we observed an extreme performance drop of several orders of magnitude for Green's method [Gre10] for higher adaptivity ratios such as 100,000:1, since this method was not designed for adaptive simulations. Additionally, Figure 4.5 (bottom right) shows a uniform simulation of comparable computational cost to the highly adaptive simulation with significantly lower visual fidelity, i.e.,the larger particles cannot move in-between the body parts of the dragon resulting in lower visual fidelity.

Non-adaptive data-structure performance: Comparing our proposed sparse data structure with a dense data structure based on [Gre10] we can see a slight overall increase in performance (see Tab. 4.1, structure "Ours" and structure "[Gre10]" for single). Due to the more ideal Morton code for particle ordering, instead of a linear ordering, we can observe an increase in performance for SPH operations. However, the re-sorting process is slowed down, as well as accessing the data structure for the construction of the neighbor-list. Embedding $\mathbb{C}$ into $\mathbb{H}$ when no hash collision occurred reduced this overhead slightly (see Tab. 4.1, structure "Embed"). The memory consumption is slightly higher as the dense structure requires $2 \cdot 153^3$ entries, whereas our structure required $2 \cdot 10^6$ entries and additional temporary arrays for construction.

Methods		Overall /ms		Re-sorting /ms		Neighbor-list /ms		Density /ms		DFSPH /ms		Memory /GiB	
Structure	Neigh-list	single	adapt	single	adapt	single	adapt	single	adapt	single	adapt	single	adapt
						Pillar scene							
[Gre10]	[WHK16]	278	2557	2.70	7.67	13.54	2067.16	1.80	1.89	216.8	316.8	1.13	6.96
Ours	[WHK16]	274	572	7.74	19.71	17.89	45.72	1.74	1.78	196.0	279.5	1.20	7.54
Embed	[WHK16]	271	543	9.76	24.78	15.67	38.77	1.72	1.76	195.2	276.9	1.20	7.54
Embed	$<$none $>$	412	1782	10.32	21.52	–	–	1.85	2.24	328.9	596.4	0.68	3.59
Embed	Histogram	253	527	9.74	22.06	12.35	26.73	2.01	1.83	196.4	285.1	1.20	7.54
Embed	Spans	291	596	10.62	21.80	7.48	28.12	0.59	3.28	222.5	302.9	1.13	5.25
Embed	Bitmask	274	866	9.97	22.39	7.80	36.41	0.67	10.64	212.9	436.9	0.91	4.42
						Dragon scene							
[Gre10]	[WHK16]	86	3185	2.40	4.34	4.71	1637.74	2.17	2.48	64.0	1342.4	0.27	7.24
Ours	[WHK16]	85	1365	7.03	11.02	6.19	35.86	2.11	2.34	57.6	1187.9	0.29	7.84
Embed	[WHK16]	84	1340	8.74	13.86	5.34	30.74	2.08	2.31	57.4	1181.8	0.29	7.84
Embed	$<$none $>$	129	4402	9.24	12.04	–	–	2.24	2.95	96.4	2547.9	0.16	3.73
Embed	Histogram	84	1302	8.72	12.43	4.37	21.93	2.43	2.43	57.7	1217.3	0.29	7.84
Embed	Spans	93	1472	9.42	12.81	2.53	22.58	0.72	4.31	65.2	1294.6	0.27	5.46
Embed	Bitmask	88	2136	8.90	12.58	2.65	28.88	0.81	14.01	62.6	1867.6	0.22	4.59

Table 4.1: The values shown here are given as the average value over 30 simulated seconds. Ours refers to the data structure presented in Sec. 4.4, with embed referring to the optimization of embedding $\mathbb{C}$ into $\mathbb{H}$. For the neighbor lists see Sec. 4.5.1 for the histogram based variant, Sec. 4.5.2 for the span based variant and Sec. 4.5.3 for the bitmask based variant. Single refers to a non-adaptive simulation and adapt refers to a simulation with an adaptive ratio of $1000 : 1$.

Adaptive data-structure performance: Looking at the adaptive results (see Tab. 4.1, structure "Ours" and structure "[Gre10]" for adapt), we can now see a significant overall difference in performance where our data structure reduces the simulation time by nearly 60%. This is mainly due to the neighbor list construction that takes about 50 times longer, requiring about half of the overall computation time per frame when using Green's method [Gre10]. We can also observe increased performance of the SPH operations, where the largest improvement is for DFSPH with a speed up of about 12% due to the improved particle ordering. However, our multiple data structure requires more memory than a dense linear structure, due to now having to store multiple data structures, however the increase is moderate at some 6-8%.

Non-adaptive Neighbor list performance: When comparing the neighborlists presented in Sec. 4.5, we observe significantly different performance between the different neighbor-list approaches for the non adaptive *Dragon* scene, if compared to the prior constrained list approach of [WHK17]. Our proposed constrained list performs almost identical with identical memory consumption. Utilizing our proposed bit mask approach, we observe a slightly slower simulation as DFSPH takes slightly longer. However the density approximation was significantly faster as the neighbor-list is significantly smaller, which benefits a simple operation more than a complex one. We see a similar effect for the span based approach, but an overall slightly lower performance. Not using a neighbor-list requires significantly less memory, and slows the overall simulation down by about 50%, with much slower complex operations.

Adaptive Neighbor list performance: Considering the adaptive *Pillars* scene, we see a different ordering of the methods. Our proposed constrained list has a slightly improved performance compared to the prior approach, which is mostly due to the faster neighbor list construction. The bitmask based approach is now significantly slower (64% overall) but requires only 59% of the memory. The span based approach performs somewhat better, i.e.,it is 13% slower but requires only 70% of the memory. Compared to [WHK17], not using a neighbor list requires only 47% of memory, but it is 237% slower.

Memory consumption: Overall, non-adaptive simulations require significantly less memory, as they require less information per particle, and as such we can simulate up to 35.5 million particles, without utilizing a neighbor-list. Using our bitmask approach, we can still simulate about 25 million particles, and using the span based and constrained methods we are able to simulate about 19 million particles. Considering the relatively low impact of the bitmask approach for non-adaptive methods, this results in an increase of 32% for the maximum number of particles, when compared to prior constrained neighbor-list approach [WHK16] without a significant drop in performance. For adaptive simulations we can simulate up to 23.5 million particles, without utilizing a neighbor-list, but the performance of this approach is too low. Using our bitmask approach we can simulate about 19 million particles, and using the span based approach about 16 million particles. Using a constrained list would only allow us to simulate about 10 million particles, which, due to the relatively low overhead of the span based approach, means that we can increase the maximum number of particles by about 60% by using our span based neighbor list, when compared to the prior constrained neighbor-list approach [WHK16]. In summary, none of the neighbor list approaches is superior to the others, i.e.,none offers the highest performance at the lowest memory consumption.

Limitations: For non-adaptive simulations our proposed data-structure only offers a very minor increase in performance, due to the better memory layout, at the cost of a slightly higher memory consumption. This increased cost, however, is only required for relatively small and bounded domains and as such not a problem in general. For adaptive-methods, we often have to adjust the resolution of particles to avoid asymmetries and, thus, severe instabilities, which reduces the overall potential performance gain. In general, smoother resolution gradients are less affected by asymmetries and allow for larger speed-ups.

4.7 Conclusions

Our contributions allow us to efficiently simulate highly adaptive simulations, with adaptive ratios beyond $1,000 : 1$, without causing performance limitations due to the underlying data structuring. Using our data structure allows us to simulate unbounded domains, where the memory consumption only scales with particle count, not resolution. In addition, by using our propose neighbor list methods we can further improve performance, or significantly reduce memory consumption allowing for higher particle counts. In the future we would like to expand our work on data structures to multi GPU systems for even larger simulations.

Chapter 5

Multi Level Memory Structures for Simulating and Rendering Smoothed Particle Hydrodynamics

Contextualization

This chapter reprints the paper "Multi Level Memory Structures for Simulating and Rendering Smoothed Particle Hydrodynamics" published in the Computer Graphics Forum [WK20] as an invited extended version of the paper reproduced in Chapter 4. This paper addresses a significant problem with spatially adaptive simulations, i.e., while the simulation of high resolution surfaces was readily possible, rendering such high resolution surfaces using extracted surfaces was computationally infeasible and on-the-fly rendering approaches were not well suited to SPH-based adaptive simulations. Conceptually, this does not improve adaptive simulations per-se, but makes them significantly more practical by reducing limitations encountered when using adaptive SPH simulations.

Conceptually this paper focuses on the application of the Multi Level Memory structure [WK19] towards anisotropic SPH models and the rendering of SPH simulations. A significant problem addressed in this scope is the computationally feasible rendering of adaptive SPH based simulations without requiring memory intensive explicit surface extractions and using only information already available during the simulation, i.e., without requiring additional memory. This process makes spatially adaptive simulations more practically usable by providing a rendering approach that can be used without reducing the maximum number of particles that can be simulated on a memory-bound system.

The initial idea of using the Multi Level Memory structure proposed before, see Chapter 4, for rendering purposes came from Rene Winchenbach and was already partially utilized during the evaluation of the prior publication. During the VMV 2019 conference discussions with many colleagues showed a significant interest in this area of application and motivated the extension towards rendering. Similarly, in this context some initial discussions also raised questions about the applicability to anisotropic SPH models, leading to their inclusion in this extended version. Andreas Kolb helped during the writing processes and provided valuable feedback during the development of the method, especially during discussions regarding the anisotropic surface model.

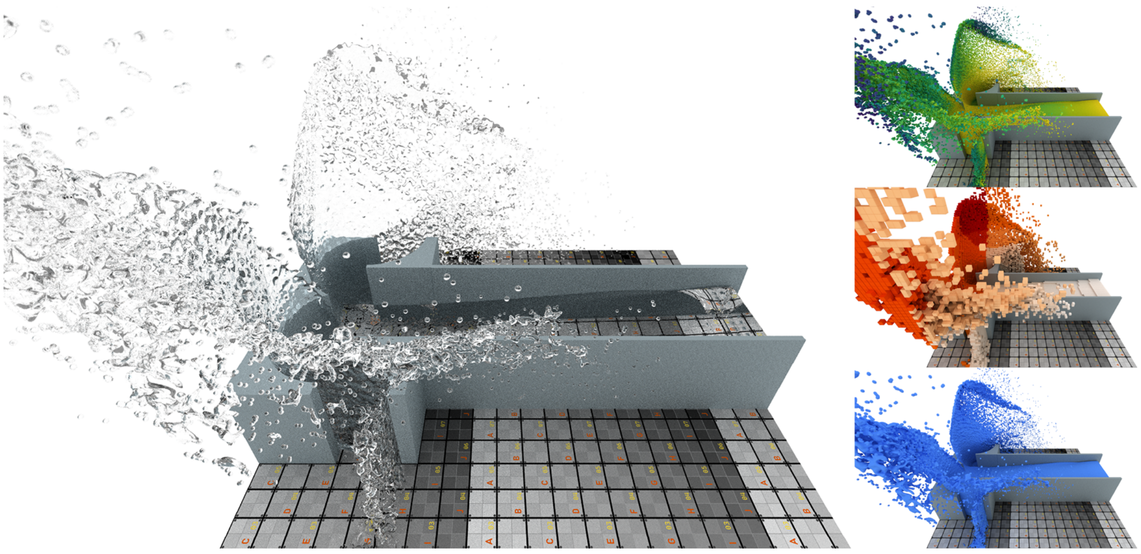


Figure 5.1: Using our rendering approach we can render an SPH fluid simulation with only a small, constant, memory overhead. Pictured here is an inlet stream colliding with an obstacle using an anisotropic surface rendering. The images to the right show the underlying particles, color coded for their velocity (top), the underlying cells, color coded for their Morton code (middle), and an opaque fluid surface (bottom).

Abstract

In this paper we present a novel hash map-based sparse data structure for Smoothed Particle Hydrodynamics (SPH), which allows for efficient neighborhood queries in spatially adaptive simulations as well as direct raytracing of fluid surfaces. Neighborhood queries for adaptive simulations are improved by using multiple independent data structures utilizing the same underlying self-similar particle ordering, to significantly reduce non-neighborhood particle accesses. Direct raytracing is performed using an auxiliary data structure, with constant memory consumption, which allows for efficient traversal of the hash map-based data structure as well as efficient intersection tests. Overall, our proposed method significantly improves the performance of spatially adaptive fluid simulations and allows for direct raytracing of the fluid surface with little memory overhead.

5.1 Introduction

Highly detailed and realistic fluid simulations have become an essential part of modern computer graphics, where Smoothed Particle Hydrodynamics (SPH) [GM77] provides a good balance of visual quality and computational cost [UHT17]. Recent advances enable highly adaptive incompressible fluid simulations [WHK17], which dedicate computational resources where they are most beneficial to the desired outcome, i.e. at the fluid surface. However, adaptive simulations with adaptivity ratios of $1000 : 1$ and higher suffer from significant performance drops due to limitations in existing underlying data structures. CPU-based SPH simulations commonly use compact hash maps [Ihm+11], which are difficult to apply on GPUs. GPU-based SPH simulations commonly use dense cell structures [Gre10; Gos+10] or linked list based structures [Dom+13; WRR18], which suffer from high memory usage and less than ideal particle orderings. Additionally, rendering the resulting fluid surfaces often involves either expensive explicit surface extraction methods [Aki+13a; Wu+17] using marching cubes [LC87], which cannot readily be done on the fly, or screen space-based approaches [LGS09; XZY17], which result in lower quality visual results and cannot readily handle refraction effects. Furthermore, many rendering methods have varying memory requirements, e.g.,depending on particle resolution and not particle count, making them impractical to use on the fly on a GPU as this requires an overly conservative maximum number of particles to not run out of memory during a simulation.

In this paper, we present a hash map-based data structure, which is specifically designed to handle the requirements of highly adaptive SPH methods, on GPUs and CPUs, and is readily extended to enable direct raytracing of the fluid surface. Our proposed data structure works by utilizing a hash map to efficiently access a compact cell list, which refers to particles sorted by a self-similar ordering. We extend this method by efficiently creating multiple distinct data structures, based on different cell sizes, by utilizing the self-similarity. Our method allows us to significantly reduce the number of non-neighbor particle accesses by providing an appropriate data structure for different particle resolutions. Furthermore, we add an auxiliary data structure to facilitate traversal of our data structure during rendering, which also enables efficient ray-fluid intersection tests. Our proposed method significantly improves the practical applicability of adaptive simulations, and substantially reduces the data structure overhead, as well as enabling direct raytracing of fluid simulations, with constant memory overhead. Finally, our method requires a constant amount of memory, regardless of simulation domain size, particle resolution, or particle distribution, allowing for unbounded simulation domains.

5.2 Related work

SPH has been an active field of research since its introduction by Gingold and Monaghan [GM77]. Whereas initially only stiff equations of state were used to simulate weakly compressible fluids [MCG03; BT07], recent advances allow both divergence-free and incompressible fluids [BK15; Gis+19], allowing for highly detailed and realistic fluid simulations. For a general overview of SPH methods we refer the reader to [Kos+19].

Spatially adaptive SPH methods using splitting and merging were initially introduced by Desbrun and Cani [DC99], however directly changing particle resolutions causes instabilities. To reduce these instabilities Adams et al. [Ada+07] adjusted particle positions after splitting, Keiser et al. [Kei+06] used virtual link particles of neighboring resolutions, Orthmann and Kolb [OK12] used temporal blending, Horvath and Solenthaler [HS13] used multiple simultaneous simulations and Winchenbach et al. [WHK17] used an additional process of mass redistribution. However, even though recent work enables adaptive ratios in excess of $10,000 : 1$, these methods are constrained by significant performance limitations due to existing data structures [WHK17].

To render fluid simulations there are three commonly used approaches : explicit surface extraction, ray casting or splatting. Explicit surface extraction methods are commonly based on marching cubes [LC87] or metaballs [Bli82], which have been adapted over time for GPUs [Wu+17; SI12], for varying spatial surface resolutions [Aki+13a], or for direct rendering [KSN08], however these methods often have highly varying memory requirements making them difficult to use on the fly. Ray casting methods are commonly found in volume rendering approaches [DCH88] where the fluid volume is commonly resampled into a 3D uniform grid [NJB07] or a perspective aligned grid [FAW10] and then rendered using standard volume ray casting [KW03]. However, these methods commonly require resampling of the full simulation data into memory intensive grids. Splatting methods [Zwi+01] used in real time applications, due to their computational simplicity, render particles as simple geometry in screen space and then smooth the resulting depth image [MSD07; LGS09; XZY17], however these methods cannot handle refractions and reflections properly. Outside of fluid simulations, various other rendering approaches exist, i.e. for point clouds [Ber+17], however they are usually not directly applicable.

For SPH-based simulations, Ihmsen et al. [Ihm+14] give a good overview of existing data structure methods for CPUs, and identify a hash map-based method [Ihm+11] as the most efficient data structure. This approach is, however, not directly applicable to GPUs due to the way in which the hash map is constructed. For GPU based simulations, Green [Gre10] introduced a method utilizing a fixed domain with linearly indexed cell lists. A similar approach was used by Dominguez et al. [DCG11], optimized for multiple GPUs. Goswami et al. [Gos+10] used Morton codes, however, their approach introduces a complex scheme to balance workloads on GPUs, making it difficult to implement and utilize. In order to limit memory usage on GPUs, Winchenbach et al. [WHK16] introduced an iterative process to constrain the size of so-called Verlet-lists, which are used to store references to neighboring particles. However, all of these methods suffer from scaling and performance problems for adaptive simulations due to excessive non-neighbor particle accesses.

Many generic data structures have been developed for computer animation, i.e. perfect hash maps to store sparse voxel data [LH06; Gar+11], which are not easily scalable to multiple resolutions, or approximate nearest neighbor methods from machine learning [AI08], which are only approximate and designed for high dimensional data. To improve the rendering speed of explicit surface methods many methods have been developed, i.e. bounding volume hierarchies (BVH) [Gun+07; Lau+09] and kd-trees [FS05], however these structures are varying in their memory requirements making them difficult to apply on the fly. Furthermore, various CPU-based approaches exist, e.g.,OpenVDB [Mus+13], but they often require significant changes to be realized on GPU-based systems. OpenVDB was realized for GPUs as GVDB, where recently, Wu et al. [Wu+18] introduced a GVDB-based data structure for FLIP-based simulations that significantly improves performance, but which is not directly applicable to SPH, as FLIP imposes significantly different requirements on the data structure, which is an integral part of the simulation itself. Overall, none of the existing data structures can be applied directly both for rendering and simulation without significant overhead.

5.3 Foundations of isotropic and anisotropic SPH

In standard SPH a quantity A at a position x is interpolated using a weighted average of spatially close particles j as [Kos+19]

$$\langle A(\mathrm{x})\rangle = \sum_j A_j \frac{m_j}{\rho_j} W(\mathrm{x}-\mathrm{x}_j, h), \tag{5.1}$$

where m denotes the mass and ρ the density of a particle and W is a kernel function. Evaluating Eqn. 5.1 at the position of a particle i then results in

$$A_i = \sum_j A_j \frac{m_j}{\rho_j} W(\mathrm{x}_{ij}, h) = \sum_j A_j \frac{m_j}{\rho_j} W_{ij}, \tag{5.2}$$

where $\mathrm{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$. The support radius h in standard SPH formulations describes an isotropic sphere with radius h, which allows one to define the kernel function W as [DA12]

$$W_{ij} = W(\mathrm{x}_{ij}, h) = C_d \frac{1}{h^d} \hat{W}(q), \tag{5.3}$$

where $q = \|\mathrm{x}_{ij}\| / h$, d being the dimensionality of the simulation, and C_d a normalization factor. For the interaction of two particleswith different support radii, i.e., due to spatil adaptivity, h can be determined as either h_i, resulting in a gather formulation, h_j, resulting in a scatter formulation, or $\frac{h_i+h_j}{2}$, resulting in a symmetric formulation. When not evaluating an SPH quantity at a particle, but at an arbitrary position x, symmetric or gather formulations cannot be directly used and, thus, the scatter formulation is commonly utilized. Here $\hat{W}$ denotes the actual kernel function, where for an SPH simulation the cubic spline kernel [Mon05]

$$\hat{W}(q) = \left[(1-q)^3\right]_+ + \left[4(0.5-q)^3\right]_+ \tag{5.4}$$

is a common choice, with $C_3 = \frac{14}{\pi}$ and $[\cdot]_+$ being $\max(\cdot, 0)$. For rendering a more simple kernel is commonly chosen, i.e. [YT13]

$$\hat{W}(q) = \left[(1-q^2)^3\right]_+, \tag{5.5}$$

with $C_3 = \frac{315}{64\pi}$. An isotropic support radius results in an equal extent of the support domain in all directions, i.e. $h = h^x = h^y = h^z$, which leads to issues on the surface of the fluid [YT13]. Instead of this isotropic formulation, one can determine an anisotropic formulation of SPH [Owe+98]

$$W_{ij} = C_d \frac{1}{|H|} \hat{W}(||H \cdot \boldsymbol{x}_{ij}||), \tag{5.6}$$

where H is the anisotropy matrix and —H— being the determinant of H. Here the isotropic variant of SPH is equivalent to $H = \frac{1}{h}\mathbb{K}$. This anisotropic formulation, however, also means that the support domain of a particle is ellipsoidal, instead of spherical, meaning the extent of the domain in all directions is not equal, i.e. $h^x \neq h^y \neq h^z$, resulting in additional challenges for neighborhood queries as this requires non-cubic cells in a data structure. Determining a gather and scatter

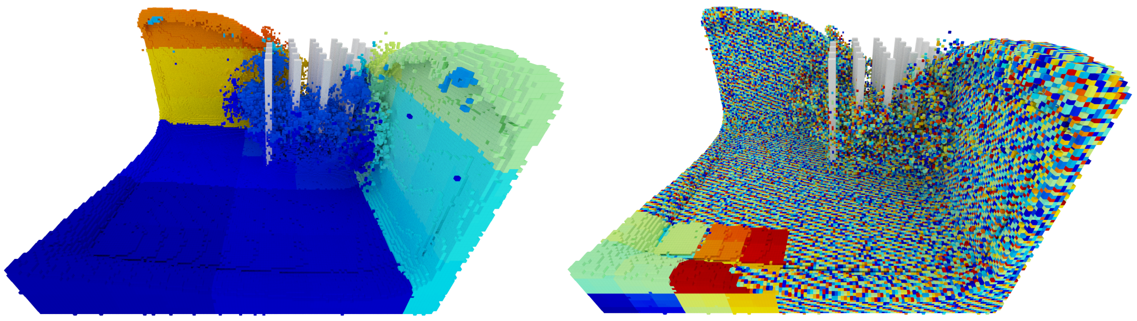


Figure 5.2: These two images show the Morton code $\mathcal{Z}$ on the left and the hashed indices $\mathcal{H}$ on the right for every occupied cell, with color coding indicating indices. The Morton code yields indices that are very similar, for spatially nearby cells, but results in significant amounts of collisions. Using a Hash function (right) yields no relation between spatial position and index, causing a low amount of collisions.

formulation for anisotropic SPH can be done directly using the anisotropic matrix of either particle, however to yield a symmetric formulations we use [HK89]

$$\hat{W}(||H \cdot \boldsymbol{x}_{ij}||) = \frac{1}{2}\left[\hat{W}(||H_i \cdot \boldsymbol{x}_{ij}||) + \hat{W}(||H_j \cdot \boldsymbol{x}_{ij}||)\right]. \tag{5.7}$$

We utilize a standard isotropic formulation for the simulation, and the anisotropic formulation only for a fluid surface rendering using [YT13]. For an isotropic formulation the support radius h_i for particle i can be determined as $h_i = \sqrt[3]{N_h V_i}$ [DA12; WHK16], with V_i being the rest volume of a particle, i.e. $\frac{4}{3}\pi r^3$ for a particle of radius r, and $N_h = 50$ for the cubic spline kernel.

5.4 Simulation data structure

The main purpose of a data structure for SPH is to relate the spatial position of a particle with its location in memory in order to reduce the number of particle accesses from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot c)$, where c is the number of particles accessed for each particle. Therefore, with $c \ll n$ this results in an asympotically linear scaling instead of quadratic scaling. One possible approach is to divide the simulation domain into uniform cells of size h [Gre10; Ihm+14], with h being the particle support radius. Note that this notion of a *cell* does not introduce any grid-based methodology into SPH and is solely for data handling. Owing to this, a particle only needs to consider at most 27 cells (a $3 \times 3 \times 3$ sub-grid) for accessing (potentially) neighboring particles. The sphere described by the support radius of a particle will, on average, contain N_h neighbors [DA12] within a volume of $\frac{4}{3}\pi h^3$, whereas the sub-grid of all potential neighbors has a volume of $27h^3$. This means that the sub-grid will contain, on average, $\frac{81}{4\pi}N_h \approx 6.5N_h$ particles, i.e.,15.5% of all potential neighbors are actual neighbors, i.e.,the factor m in $\mathcal{O}(n \cdot m)$ becomes 325. For an adaptive ratio of $1000:1$, however, only 0.016% of all considered particles are neighbors as a cell of the same size would now contain $\frac{81000}{4\pi}N_h$ particles, causing significant performance problems [WHK17]. We are going to introduce our general data structure for non-adaptive simulations in Sec. 5.4.1, and the changes required for adaptive simulations in Sec. 5.4.2. Sec. 5.4.3 will introduce our data structure for rendering and Sec. 5.4.4 will discuss how we can optimize the memory layout of our data structure.

5.4.1 Single-level data structure

Isotropic SPH methods commonly use cubic cells for data handling [Gre10] [Ihm+11], as the support domain of a particle extends equally along all axes. Anisotropic SPH methods require non-cubic cells for data handling, as the support domain of a particle might extend differently along all axes. As our overall method uses both formulations, we determine the effective support extent $\mathbf{h}$ for each particle, based on the isotropic support radius h_i^i and the anisotropic support extent along all axes $[h_i^x, h_i^y, h_i^z]$, as

$$\mathbf{h}_i = \left[\max\left(h_i^i, h_i^x\right), \max\left(h_i^i, h_i^y\right), \max\left(h_i^i, h_i^z\right)\right]. \tag{5.8}$$

Alternatively, anisotropic SPH methods can be treated as isotropic, for data handling, using $h_i^i = \max(h_i^i, h_i^x, h_i^y, h_i^z)$, however, this places more particles in each cell, causing more non-neighbor accesses in total. The cell size C_{max} is set to the same value as the largest support extent of all particles. This ensures that all neighbors for a particle are contained in a $3 \times 3 \times 3$ sub-grid, which would not be possible for an arbitrary cell size. As such, we calculate C_{max} as

$$\mathrm{C}_{\text{max}} = \max\{\mathbf{h}_0, ..., \mathbf{h}_{n-1}\}. \tag{5.9}$$

The simulation domain itself is similarly determined as the axis aligned bounding box, from $\boldsymbol{D}_{\text{min}}$ to $\boldsymbol{D}_{\text{max}}$, surrounding the positions of all particles. We determine these bounds by using reduction operations over all particle positions $\boldsymbol{x}_i$

$$\boldsymbol{D}_{\text{min}} = \min\{\boldsymbol{x}_0, ..., \boldsymbol{x}_{n-1}\}, \boldsymbol{D}_{\text{max}} = \max\{\boldsymbol{x}_0, ..., \boldsymbol{x}_{n-1}\}. \tag{5.10}$$

These bounds are used to calculate the size of the simulation domain in cells as

$$\boldsymbol{D} = \left\lceil \frac{\boldsymbol{D}_{\text{max}} - \boldsymbol{D}_{\text{min}}}{\mathrm{C}_{\text{max}}} \right\rceil. \tag{5.11}$$

When using dense data structures, $\boldsymbol{D}$ needs to be kept constant to avoid reallocating memory when particles move outside the current simulation domain. This, in turn, limits the scene's extend as it needs to be known a-priori. We then calculate the integer coordinates $\bar{\boldsymbol{x}}$ for any position $\boldsymbol{x}$ based on the lower simulation bound $\boldsymbol{D}_{\text{min}}$ and the cell size C_{max} as

$$\bar{\boldsymbol{x}} = \left\lfloor \frac{\boldsymbol{x} - \boldsymbol{D}_{\text{min}}}{\mathrm{C}_{\text{max}}} \right\rfloor. \tag{5.12}$$

This can be used to determine a linear index $\mathcal{L}$ as

$$\mathcal{L}(\bar{\boldsymbol{x}}) = \bar{\boldsymbol{x}}_x + \boldsymbol{D}_x\left(\bar{\boldsymbol{x}}_y + \boldsymbol{D}_y\left(\bar{\boldsymbol{x}}_z\right)\right), \tag{5.13}$$

where the subscript denotes the dimension. In a dense cell grid, we can utilize $\mathcal{L}(\bar{\boldsymbol{x}})$ to find the memory location of any position in space. Dense data structures, however, are not desirable as their memory consumption scales with both the simulation domain $\boldsymbol{D}$ and the cell size C_{max}, instead of scaling with the particle count $n_{\text{particles}}$. The Morton code [Mor66], also sometimes referred to as the Z-ordering, is an alternative indexing scheme, which describes a self-similar space-filling curve. We can determine $\mathcal{Z}(\bar{\boldsymbol{x}})$ by interleaving the binary representation of an integer coordinates as

$$\bar{\boldsymbol{x}} = \begin{pmatrix} ...\bar{\boldsymbol{x}}_x^3\bar{\boldsymbol{x}}_x^2\bar{\boldsymbol{x}}_x^1\bar{\boldsymbol{x}}_x^0 \\ ...\bar{\boldsymbol{x}}_y^3\bar{\boldsymbol{x}}_y^2\bar{\boldsymbol{x}}_y^1\bar{\boldsymbol{x}}_y^0 \\ ...\bar{\boldsymbol{x}}_z^3\bar{\boldsymbol{x}}_z^2\bar{\boldsymbol{x}}_z^1\bar{\boldsymbol{x}}_z^0 \end{pmatrix} \rightarrow \mathcal{Z}(\bar{\boldsymbol{x}}) = ...\bar{\boldsymbol{x}}_z^3\bar{\boldsymbol{x}}_y^3\bar{\boldsymbol{x}}_x^3\bar{\boldsymbol{x}}_z^2\bar{\boldsymbol{x}}_y^2\bar{\boldsymbol{x}}_x^2\bar{\boldsymbol{x}}_z^1\bar{\boldsymbol{x}}_y^1\bar{\boldsymbol{x}}_x^1\bar{\boldsymbol{x}}_z^0\bar{\boldsymbol{x}}_y^0\bar{\boldsymbol{x}}_x^0,$$

where the superscript denotes a specific bit. Using a 32 bit Morton code results in 10 bit per dimension, meaning each dimension contains a maximum of $\#K = 1024$ cells. A 64 bit Morton code results in 21 bit per dimension, meaning a maximum of $\#K = 2097152$ cells per dimension. On one hand it would be possible to create an octree directly from Morton codes [Kar12], as this code represents the ordering of an octree. To find neighboring particles, in SPH, we only have to consider a small spatial region and, as such, many octree notes, e.g.,the root node, contain no useful information but still require memory. Furthermore, traversing an octree is computationally relatively expensive and the memory consumption of an octree is not independent of the content. On the other hand, a dense data structure using a Morton code would require excessive amounts of memory, especially as a 64 bit Code would require 2^{60} entries.

We instead propose to create a list of all occupied cells, as the number of occupied cells n_{occupied} is bound by the number of particles $n_{\text{particles}}$, as the worst case would be every particle occupying a different cell. Morever, the lower bound of occupied cells is based on the number of particles per cell, which is bounded by incompressibility, of $\frac{3}{4\pi}N_h$, see Sec. 5.4, resulting in approximately 12 particles per cell for the cubic spline kernel, i.e.,$n_{\text{occupied}} = \frac{1}{12}n_{\text{particles}}$. However, during a simulation many cells contain less particles, i.e. particles flying through the air after an impact, resulting in an average ratio of approximately $1:6$ over the course of a simulation.

To generate the list of occupied cells, we first re-sort all particles according to their Morton code $\mathcal{Z}_i = \mathcal{Z}(\bar{\boldsymbol{x}}_i)$. Using this ordering we create a list $\mathbb{C}$ of length $n_{\text{particles}} + 1$, where each element is determined as

$$\mathbb{C}[i] = \begin{cases} i & \text{, if } i = 0 \vee \mathcal{Z}_i \neq \mathcal{Z}_{i-1} \\ -1 & \text{, if } \mathcal{Z}_i = \mathcal{Z}_{i-1} \\ n_{\text{particles}} & \text{, else.} \end{cases} \tag{5.14}$$

$\mathbb{C}$ now contains either a marker entry (-1 or $n_{\text{particles}}$), or the first index of a particle in an occupied cell, which is similar to the approach by Green [Gre10]. We can now compact $\mathbb{C}$, by removing all invalid entries, which gives us a list $\mathbb{C}_{\text{compact}}^{\text{begin}}$ of length $n_{\text{occupied}} + 1$. Using this list of occupied cell beginnings, we can calculate the number of particles in each occupied cell as

$$\mathbb{C}_{\text{compact}}^{\text{length}}[i] = \mathbb{C}_{\text{compact}}^{\text{begin}}[i+1] - \mathbb{C}_{\text{compact}}^{\text{begin}}[i]. \tag{5.15}$$

This compact list, however, does not yield any way to find the memory location for a particle based on its spatial location. To resolve this, we propose to apply a hash

ALGORITHM 5.1: The algorithm to access the cell associated with an arbitrary integer coordinate using our proposed sparse data structure without embedding $\mathbb{C}$ into $\mathbb{H}$. Note that an empty cell can map to the same hash map entry as an occupied cell, without causing a collision, and as such we always check $\mathcal{Z}_c = \mathcal{Z}_j$ to avoid returning a wrong cell.

```
1  Calculate $\bar{\boldsymbol{x}}_c$ for the cell we are looking for
2  Calculate $\mathcal{Z}_c$ and $\mathcal{H}_c$ for $\bar{\boldsymbol{x}}_c$
3  Look-up $b = \mathbb{H}^{\text{begin}}[\mathcal{H}_c]$ and $l = \mathbb{H}^{\text{length}}[\mathcal{H}_c]$
4  If $l \neq 0$
5      For $h \in [b, b+l)$
6          Look-up Particle $j = \mathbb{C}^{\text{begin}}_{\text{compact}}[h]$
7          Calculate $\bar{\boldsymbol{x}}_j$ and $\mathcal{Z}_j$
8          If $\mathcal{Z}_c = \mathcal{Z}_j$
9              Return $\mathbb{C}_{\text{compact}}[b]$
10 Return not found
```

map on top of $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and $\mathbb{C}^{\text{length}}_{\text{compact}}$. Depending on the intended purpose of the data structure, i.e. solely simulation or simulation and rendering, different hash functions should be used. The cell information is only accessed once during the simulation, as it is only required for the creation of a neighbor list, but accessed many times during rendering, as rendering requires evaluating SPH estimates at arbitrary positions, which have no neighbor list associated. As such, for the simulation a hash function with few collisions is preferable, but for rendering a hash function with good spatial locality is preferable. If the hash map is only required for the simulation we use the hash function of Teschner et al. [Tes+03], which is determined using three large prime numbers $p_1 = 73856093$, $p_2 = 19349663$, $p_3 = 83492791$ and the size of the hash table n_{hash} as

$$\mathcal{H}(\bar{\boldsymbol{x}}) = (p_1\bar{\boldsymbol{x}}_x + p_2\bar{\boldsymbol{x}}_y + p_3\bar{\boldsymbol{x}}_z)\,\%n_{\text{hash}}. \tag{5.16}$$

We choose n_{hash} as the smallest prime number larger than the maximum number of particles in a simulation, as this results in a relatively sparse hash map with few collisions, in general. Fig. 5.2 shows a comparison of this hash function with the Morton code, which demonstrates the lack of spatial locality. If the hash map is also used for rendering we use a more simple hash function, directly based on the Morton code, as

$$\mathcal{H}(\bar{\boldsymbol{x}}) = \mathcal{Z}(\bar{\boldsymbol{x}})\%n_{\text{hash}}, \tag{5.17}$$

which will result in more hash collisions, but also much greater cache locality. Alternatively other hash functions could be used, i.e. perfect hash functions[LH06], but they are more expensive to create and especially more computationally expensive to evaluate, making them unattractive for rendering. In general, we place the cell information in the hash map, if there were no collisions in this hash map entry, as this avoids one level of indirection.

The hash map itself is similar to the cell list in that it contains a begin entry and a length entry, where the begin entry now points to the first cell mapped to a hash table entry and the length entry indicates how many cells map to this hash table entry. If there is no cell then the length is 0, if there is a single cell occupying

ALGORITHM 5.2: Our proposed single resolution data-structure algorithm. This algorithm first re-sorts all particles and then creates a compact cell table followed by the creation of our hash map as described in Section 5.4.1.

```
1  Initialize
2      Calculate $\mathrm{C}_{\text{max}}$, $\mathbf{D}_{\text{min}}$ and $\mathbf{D}_{\text{max}}$ using reductions
3      Calculate $P^2$ based on $\boldsymbol{D}$
4      Re-sort particles using $\mathcal{Z}^{\text{fine}}$
5  Cell table creation
6      Initialize $\mathbb{C} = -1$ and $\mathbb{C}^{\text{length}}_{\text{compact}} = 0$
7      Create $\mathbb{C}$ based on Morton codes of consecutive particles
8      Compact $\mathbb{C}$ into $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and determine $\mathbb{C}^{\text{length}}_{\text{compact}}$
9      Calculate $\mathcal{H}_i$ for all particles
10     Re-sort $\mathbb{C}^{\text{begin}}_{\text{compact}}$ and $\mathbb{C}^{\text{length}}_{\text{compact}}$ based on the hash index of the first contained
         particle.
11 Hash map creation
12     Initialize $\mathbb{H}^{\text{begin}} = -1$ and $\mathbb{H}^{\text{length}} = 0$
13     Create $\mathbb{H}^{\text{begin}}$ based on compacted cell list
14     Calculate $\mathbb{H}^{\text{length}}$
15     Embed $\mathbb{C}_{\text{compact}}$ into $\mathbb{H}$ if $\mathbb{H}^{\text{length}} = 1$
```

this hash map the length is 1 and a length > 1 indicates a hash collision. The hash map, contrary to the cell list, is not compacted and as such allows us access via the hash index of an integer coordinate $\mathcal{H}(\bar{\boldsymbol{x}})$. The process required to find a specific cell c based on the cells integer coordinates $\bar{\boldsymbol{x}}_c$ is described in Algorithm 5.1.

To create the hash table $\mathbb{H}$ we first start by initializing all hash table entries as invalid, i.e.,0 length, and re-sort the list of occupied cells according to the hashed index of the first particle in this cell. We then, for each occupied cell i, set

$$\mathbb{H}^{\text{begin}}[\mathcal{H}_i] = i, \text{ if } i = 0 \vee \mathcal{H}_i \neq \mathcal{H}_{i-1}, \tag{5.18}$$

where we then set the length entry, for each occupied cell i, as

$$\mathbb{H}^{\text{length}}\left[\mathcal{H}_i\right] = i - \mathbb{H}^{\text{begin}}\left[\mathcal{H}_i\right] - 1, \text{ if } i = n_{\text{occupied}} \vee \mathcal{H}_i \neq \mathcal{H}_{i+1} \tag{5.19}$$

which naturally handles hash collisions as the predicate is based on $\mathcal{H}_i \neq \mathcal{H}_{i+1}$ which is only true for the last cell associated with a certain hash value. The overall algorithm to create our hash map based data structure, for a single level, is described in Algorithm 5.2.

5.4.2 Multi-level data structures

The prior section described our approach for a single fixed cell size, which would suffer from the same problems for adaptive simulations as prior methods, due to a mismatch of cell size and particle resolution. Utilizing the self-similarity of the underlying Morton code, however, we can efficiently create multiple, distinct, data structures for different cell sizes on the same underlying particle data. All power of 2 times coarser resolutions follow the same underlying ordering, due the octree-like structure of Morton codes. The lowest required resolution for the data structure is still $\mathbf{C}_{\text{max}}$, resulting in a coarse grid size of $\mathbf{D}_{\text{c}}$; see Eqn. 5.11. Using the largest component of the grid, $P = \max\left(\boldsymbol{D}_{\text{c}}^{x}, \boldsymbol{D}_{\text{c}}^{y}, \boldsymbol{D}_{\text{c}}^{z}\right)$, we determine the smallest cell size possible, as the bit length of the Morton code used limits the number of cells per dimension to $\#K$; see Sec. 5.4.1, as

$$C_{\text{fine}} = C_{\text{max}} \frac{2^{\lceil \log_2(P) \rceil}}{\#K}. \tag{5.20}$$

Here $2^{\lceil \log_2(P) \rceil}/\#K = 2^{-L_{\text{fine}}}$, $L_{\text{fine}} \in \mathbb{N}$ is the refinement factor. The algorithm described in the Sec. 5.4.1 can now be extended by creating the cell list and hash map for a Morton code $\mathcal{Z}^{\text{max}}$ based on C_{max} and additional finer levels $0 < L \leq L_{\text{fine}}$ using the integer coordinates

$$\bar{\boldsymbol{x}}^{L} = \left\lfloor \frac{\boldsymbol{x} - \boldsymbol{D}_{\text{min}}}{C_{\text{max}} \cdot 2^{-L}} \right\rfloor. \tag{5.21}$$

$L = 0$ results in the same data structures as the single-level version of this algorithm and $L = L_{\text{fine}}$ the finest possible data structure, with the given Morton code. The corresponding Morton code is determined as $\mathcal{Z}^{L}(\boldsymbol{x}) = \mathcal{Z}(\bar{\boldsymbol{x}}^{L})$. We relate the maximum level L_{max} to the maximal adaptivity ratio α of the simulation as

$$L_{\text{max}} = \min\left\{ \left\lceil log_2 \sqrt[3]{\alpha} \right\rceil, L_{\text{fine}} \right\}, \tag{5.22}$$

and generate the data structure for all levels $0 \leq L \leq L_{\text{max}}$. We store all data structures within single continuous arrays, which allows us to calculate the hashed indices by simply adding an offset based on the level to any hash function

$$\mathcal{Z}^{L}(\bar{\boldsymbol{x}}) = L n_{\text{hash}} + \mathcal{Z}(\bar{\boldsymbol{x}}). \tag{5.23}$$

Furthermore, we determine the appropriate level for a particle i according to its support radius as

$$L_i = \operatorname{clamp}\left(\left\lfloor -\log_2 \frac{h_i}{C_{\text{max}}} \right\rfloor, 0, L_{\text{fine}} - 1 \right). \tag{5.24}$$

Thus, every particle can easily access all neighbors at any scale $L \in [0, L_{\text{max}}]$ using the data structure for L. However, when a particle i only looks for neighbors at its level L_i, we may encounter asymmetric interactions, as neighbor searches are limited by the cell size for level L_i; see Fig. 5.3.

More formally, asymmetric interactions occur when a particle i of lower level L_i is interacting with a particle k of a higher level L_k, i.e.,if $L_k > L_i \wedge x_{ik} < h_{ik}$. In order to resolve the asymmetry, we iterate over all neighboring particles k of i and determine their integer coordinate distance, for the cell size associated with L_k as $\bar{\boldsymbol{x}}_{ik}^{L_k} = \bar{\boldsymbol{x}}_{i}^{L_k} - \bar{\boldsymbol{x}}_{k}^{L_k}$. We use $\bar{\boldsymbol{x}}_{ik}^{L_k}$ to identify the problem case that k does not see particle i by checking

$$\left\| \bar{\boldsymbol{x}}_{ik}^{L_k} \right\|_1 = \max(|\bar{\boldsymbol{x}}_{ik,x}^{L_k}|, |\bar{\boldsymbol{x}}_{ik,y}^{L_k}|, |\bar{\boldsymbol{x}}_{ik,z}^{L_k}|) > 1. \tag{5.25}$$

We then resolve this issue by atomically updating L_k to be at least L_i, as this ensures that k will search distant enough cells to find i. The overall changes to the single resolution algorithm are relatively minor but are outlined in Algorithm 5.3.

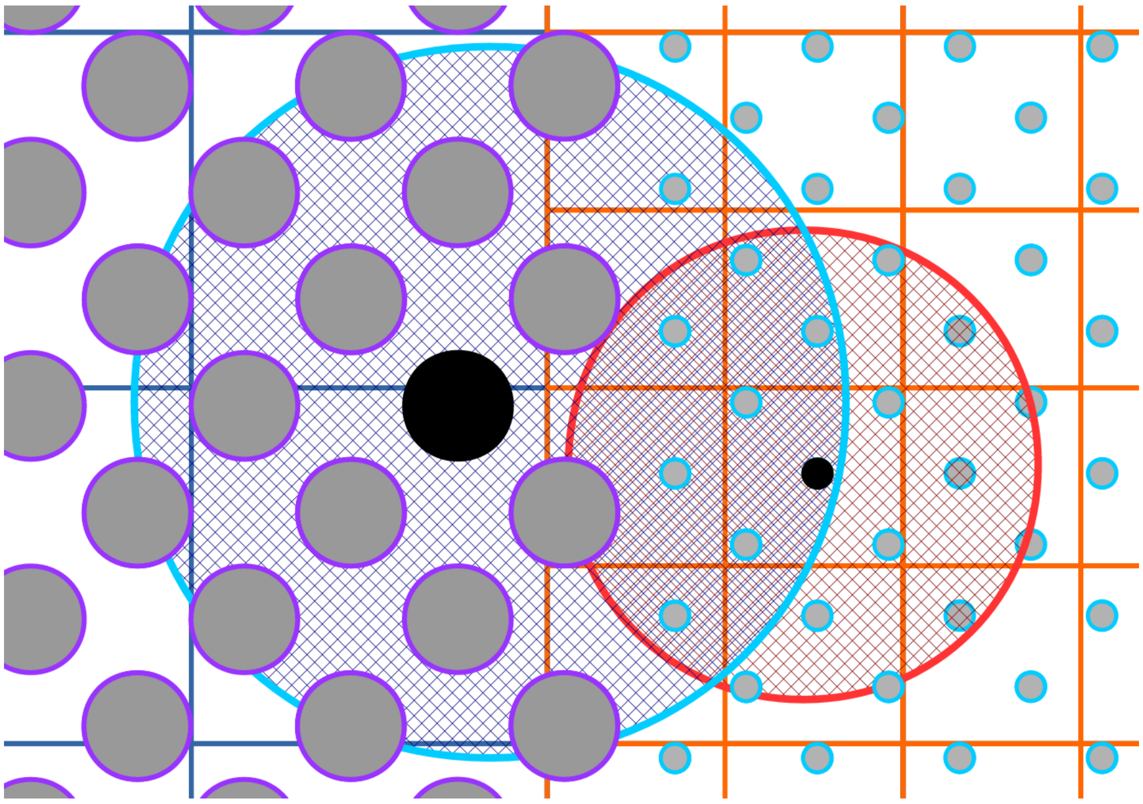


Figure 5.3: Asymmetry interaction: Two particles at different levels may not mutually see each other due to the different cell size. Here, the lower level particle to the left sees the higher level particle to the right, but not vice versa.

5.4.3 Rendering data structure

A naïve way to render the fluid surface using ray-casting would be to treat the underlying cell structure as a grid and use traditional ray-casting approaches, i.e. [KW03]; however, this would require checking every cell inside of the simulation domain along a ray for potential ray-fluid intersections, even though most cells contain no actual fluid surface. Alternatively, simply checking for fluid-ray intersections within occupied cells does not work as the fluid surface of particles within one cell extends into other cells; see Fig. 5.4. At worst, the fluid surface for every single particle could be distributed amongst 27 cells, which would require a total of $27 \cdot n_{\text{particles}}$ cells; however, explicitly storing these cells is not practical, due to limited memory resources on GPUs. Instead, we propose to store this information only implicitly using a hash map, as it suffices to mark whether any cell, that potentially contains fluid surface, maps to a certain hash map entry. Morever, cells that contain no fluid particles, themselves, can still contain fluid surfaces. Thus using the data-structure directly require checking every cell intersected by a ray, instead of only checking cells that can contain fluid surfaces.

The main purpose of this data structure is not to find particles close to a spatial location, as was the case for the simulation data structure, but instead to check whether a cell at a spatial location could contain fluid surface. Instead of storing references to particle ranges contained in a cell, we only store the Morton code of a cell, as many cells that could contain fluid surface do not contain any particles. The hash map used here uses the same size as the one for the simulation, which,

ALGORITHM 5.3: Changes required to create multiple levels of our data structure

```
1 Initialize
2     Calculate $\mathrm{C}_{\text{max}}$, $\mathbf{D}_{\text{min}}$ and $\mathbf{D}_{\text{max}}$ using reductions
3     Calculate $P^2$ based on $D$
4     Re-sort particles using $\mathcal{Z}^{\text{fine}}$
5     Calculate Ideal level $L_i$ for every particle
6 For every Multi-Level-Memory level $L$
7     Execute Cell table creation and Hash map creation using $\mathcal{Z}^L$ and $\mathcal{H}^L$
        respectively.
8 Finalize
9     Enforce symmetric interactions
```

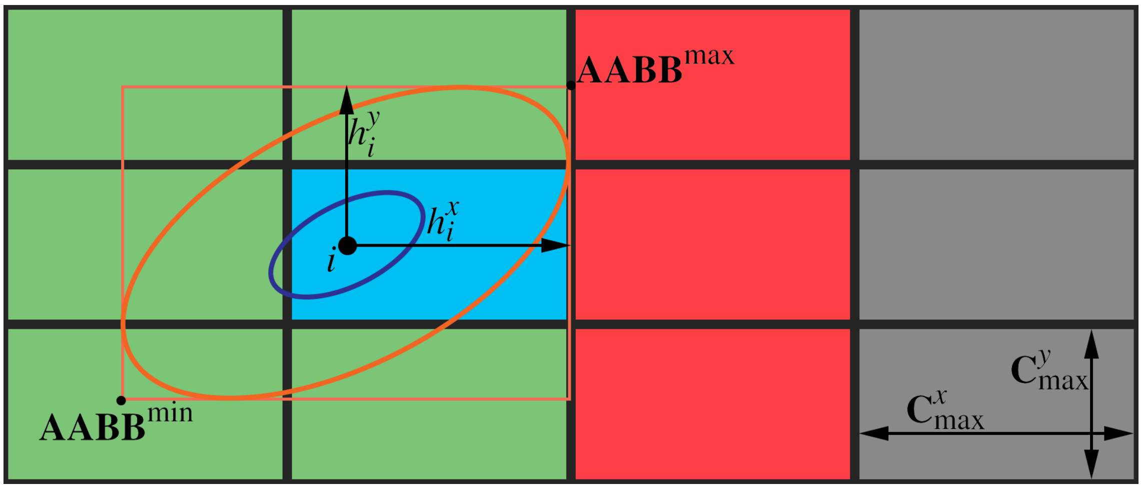


Figure 5.4: An anisotropic particle in a non-cubic grid. The blue cell contains the actual particle position, red cells are neighboring but cannot contain fluid surface, green cells are neighboring and can contain fluid surface, and gray cells are not neighboring. The fluid surface due to the particle is colored dark blue.

due to the low number of cells containing fluid surface, results in only a moderate number of collisions, even when using non-ideal hash functions.

In order to find cells containing particles at the fluid surface, we first calculate the signed surface distance ϕ for all particles, using the method of Horvath and Solenthaler [HS13]. A particle that is close enough to the surface, i.e. $\phi_i < r_i$, is marked as a surface particle. We then iterate over all cells in the compact list of occupied cells $\mathbb{C}_{\text{compact}}$ and create a list of surface cells

$$\mathbb{R}[i] = \begin{cases} \mathcal{Z}(\mathbf{x}_{c_i^0}), & \text{if } \exists c \text{ marked as surface } : c \in \mathcal{C}_i, \\ -1, & \text{else,} \end{cases} \tag{5.26}$$

where $\mathcal{C}_i$ is the set of particles contained in cell i and c_i^0 is the first particle contained in a cell, i.e. $c_i^0 = \mathbb{C}_{\text{compact}}^{\text{begin}}[i]$. Next, we use a compact operation on this list to yield a compacted list of cells containing surface particles, denoted as $\mathbb{F}$. Next, similar to the simulation data structure, we sort this list according to the hash function and create a hash table $\mathbb{H}_R$, as before, using this sorted compact list. Next, we embed the cell information in the hash table, if there was no collision for a hash entry, as this avoid one level of indirection. At the end of this process,

ALGORITHM 5.4: Our proposed rendering data-structure algorithm; see Section 5.4.3.

```
Initialize
    Determine signed surface distance $\phi$ for all particles [HS13]
    Set $s[i] = 1$ if particle $i$ close to surface, otherwise $s[i] = 0$
Find surface particle cells
    Determine cells containing surface particles as $\mathbb{R}$; see Eqn. 5.26
    Compact cells containing surface particles into $\mathbb{F}$
    Sort $\mathbb{F}$ based on hash function using stored Morton codes
    Create $\mathbb{H}_R^{\text{begin}}$ based on sorted list
    Calculate $\mathbb{H}_R^{\text{length}}$
    For all hash entries $h$ with $\mathbb{H}_R^{\text{length}}[h] = 1$
        Embed Store Morton code of corresponding cell directly in $\mathbb{H}_R[h]$
        Mark entry $h$ as particle cell
    For all hash entries $h$ with $\mathbb{H}_R^{\text{length}}[h] \geq 1$
        Mark entry $h$ as particle cell collision
Spread surface cells
    Calculate $\mathbf{AABB}_c^{\text{min}}$ and $\mathbf{AABB}_c^{\text{max}}$ for all cells $c \in \mathbb{F}$
    For each neighboring cell $n$ of all cells $c \in \mathbb{F}$
        If $n$ overlaps AABB of $c$
            Calculate Morton code $\mathcal{Z}_n$ and hash key $\mathcal{H}_n$ for cell $n$
            If $\mathbb{H}_R[\mathcal{H}_n]$ empty
                Store $\mathcal{Z}_n$ in $\mathbb{H}_R[\mathcal{H}_n]$
                Mark $\mathbb{H}_R[\mathcal{H}_n]$ as potential surface cell
            Else If $\mathbb{H}_R[\mathcal{H}_n]$ does not contain $\mathcal{Z}_n$ already
                Mark $\mathbb{H}_R[\mathcal{H}_n]$ as collision
```

a hash table entry either contains a cell, represented as a Morton code, or a range of cells that map to this hash entry. We then calculate an AABB for every cell $i \in \mathbb{F}$, based on the contained particles, as

$$\mathbf{AABB}_i^{\text{min}} = \min_{c \in \mathcal{C}_i} \mathbf{x}_c - \mathbf{h}_c; \mathbf{AABB}_i^{\text{max}} = \max_{c \in \mathcal{C}_i} \mathbf{x}_c + \mathbf{h}_c, \tag{5.27}$$

where $\mathbf{h}$ is the extent of support along the simulation axes. Finally, for every cell $i \in \mathbb{F}$ we check if any of the surrounding cells N_i overlap the bounding box of i. If we find an overlap with cell $n \in N_i$, we check if $n \in \mathbb{F}$ using $\mathbb{H}^R$, in which case nothing needs to be done. If $n \notin \mathbb{F}$ and the corresponding hash map entry is empty, we set the entry to indicate n as potentially having fluid surface. If the hash map entry was already occupied we mark the entry as having a collision, which means that any cell mapping to this hash entry is assumed to have fluid surface in it. This process is also described in Algorithm 5.1.

Given a position $\mathbf{x}$, with a corresponding cell q, we can query this data structure using the corresponding Morton code $\mathcal{Z}_q$ and hash key $\mathcal{H}_q$. If $\mathbb{H}_R[\mathcal{H}_q]$ is empty, no fluid surface can be contained in q. If $\mathbb{H}_R[\mathcal{H}_q]$ is marked as *collision*, q potentially contains fluid surface. Otherwise, if $\mathbb{H}_R[\mathcal{H}_q]$ contains $\mathcal{Z}_q$ then q contains particles at the fluid surface. This rendering data structure is also described in Algorithm 5.4.

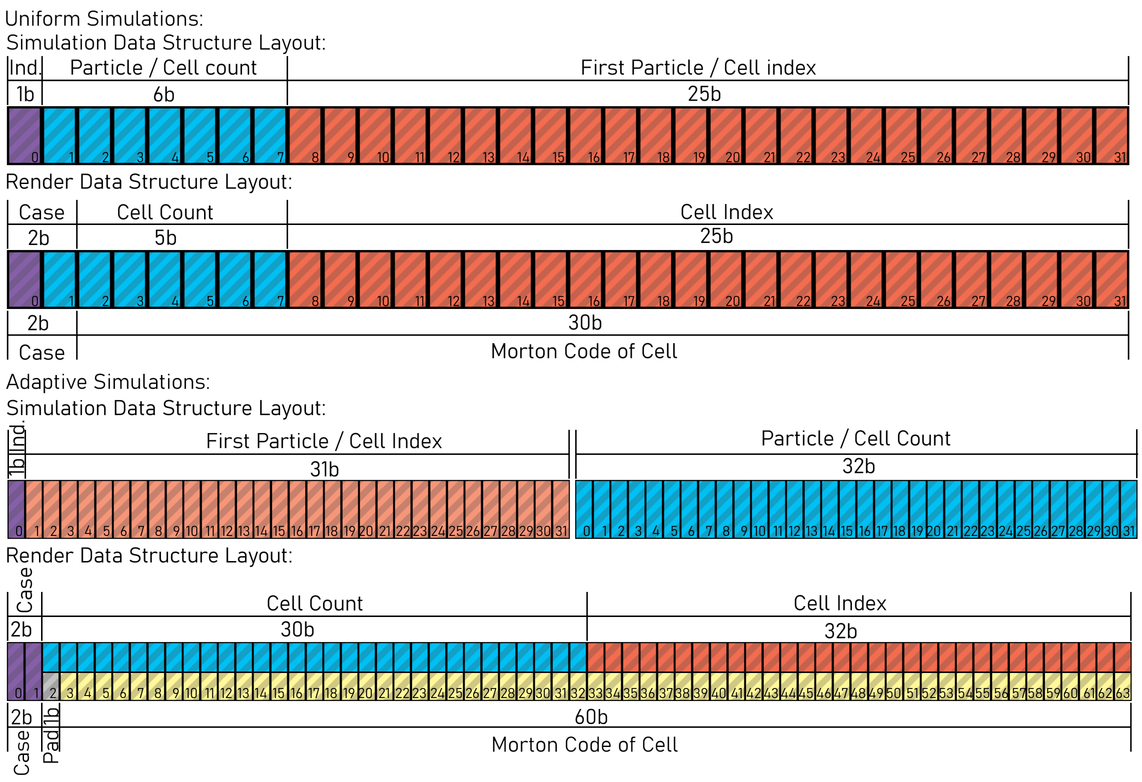


Figure 5.5: Memory layout of the data structures for simulation and rendering for uniform and adaptive simulations. The render data layout uses a union indicated by the split fields.

5.4.4 Data structure memory layout

To reduce the memory consumption, and improve the performance, of our data structure several optimizations can be made to the memory layout of the data structure. These optimizations are based on a word size of 4Byte, which reduces the number of required atomic operations significantly. For the Morton code we use 30 bits in non-adaptive simulations and 60 bits in adaptive simulations.

A hash table and cell list entry of the simulation data structure contain both a beginning and length entry, using a 4Byte integer each. To embed a cell list entry in the hash map we reserve a single bit of the beginning entry, as an indicator for the kind of data, which limits the maximum number of particles to at most $2^{31}-1$, which should not cause problems on single GPUs.

As stated at the beginning of Sec. 5.4, an average cell contains only $\frac{3}{4\pi}N_h \approx 11.94$ particles, when using a cubic spline kernel, in non adaptive simulations. Furthermore, the maximum number of particles we can simulate on a single GPU is below $32M$. As such, we only require 25 bits to represent the cell begin entry and approximately 3.6 bits to represent the number of particles per cell. As we still require a single bit to indicate if a cell entry is embedded, we chose a 25 bit integer for the cell begin entry and a 6 bit integer for the cell length entry; see Fig. 5.5. To account for larger support radii for rendering, which might increase the number of particles per cell significantly, we keep track of the actual number of particles in each cell, in a separate list, where a length entry of 63 in a cell indicates a required additional lookup. The data structure for rendering needs to indicate four cases, requiring 2 bits, as well as store the Morton code, requiring 30 bits in non

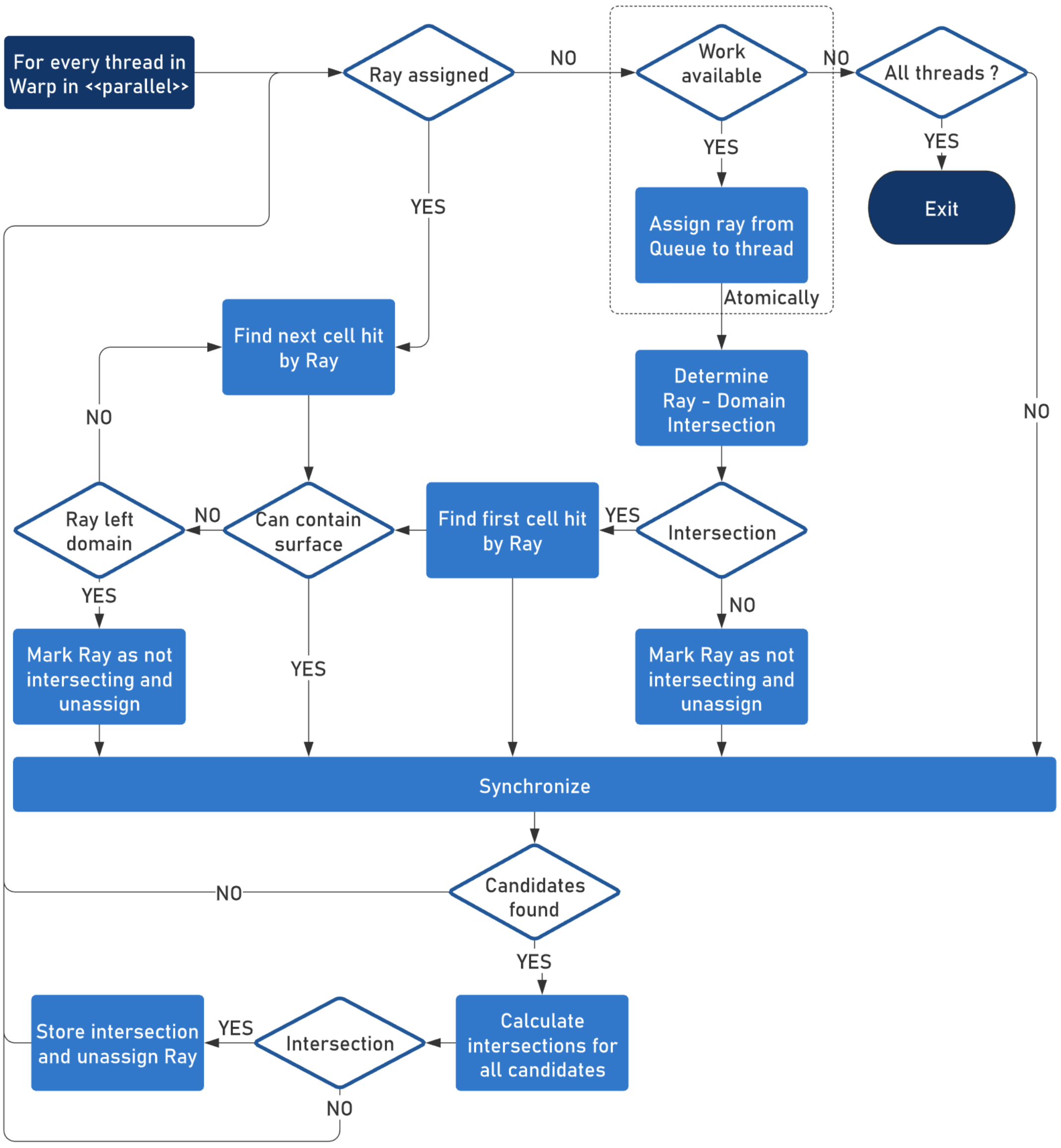

For every thread in Warp in <<parallel>>
Ray assigned
NO
Work available
NO
All threads ?
YES
Exit
YES
Assign ray from Queue to thread
Atomically
YES
Find next cell hit by Ray
NO
Determine Ray - Domain Intersection
NO
Ray left domain
NO
Can contain surface
Find first cell hit by Ray
YES
Intersection
YES
NO
Mark Ray as not intersecting and unassign
YES
Mark Ray as not intersecting and unassign
Synchronize
NO
Candidates found
YES
Store intersection and unassign Ray
YES
Intersection
Calculate intersections for all candidates
NO

Figure 5.6: The warp program used to calculate ray fluid intersections. Warp here refers to a group of threads, i.e. 32, executing in parallel on either a GPU or CPU.

adaptive simulations. As such, we can store the Morton code for a cell, and the case indicator, within a single $4B$ entry. To account for the 2 bits for the cases a hash entry in this data structure consists of a 25 bit integer for the begin entry and a 5 bit integer for the length entry, limiting the number of cells mapped to the same hash entry to 32, which should not be exceeded in practice. To embed cell information into the hash map we utilize a union, with a case field indicating the kind of entry; see Fig. 5.5

5.5 Rendering

Intersecting a ray with the fluid surface could, naïvely, be done by simply evaluating a function at every point along the ray and finding the closest value that matches the desired iso value; however, this is not practically possible. Instead, we propose to first find rays that can potentially intersect the fluid, i.e. rays intersecting the simulation domain. Next, we iterate along those rays, over the cells in the data structure, to find cells that can potentially contain fluid surfaces. Finally, we evaluate the ray-fluid intersection, but only within an intersected cell; see Fig. 5.6. Given a ray

$$\text{ray}_i(\lambda) = \mathbf{x}_i^{\text{o}} + \lambda \cdot \mathbf{d}_i, \lambda \in \mathbb{R}_0^+, \tag{5.28}$$

starting at $\mathbf{x}_i^{\text{o}}$ with direction $\mathbf{d}_i$, we check for an intersection of a ray with the simulation domain using a standard AABB intersection test. Next, in case of an intersection, we calculate the first cell hit by the ray and traverse the domain, along the ray; see Sec. 5.5.1, until either the ray exits the simulation domain, or we find a cell that can potentially contain fluid surfaces, see Sec. 5.4.3. For a found cell we then calculate an intersection using either the cells directly; see Sec. 5.5.2, treating particles as spheres; see Sec. 5.5.3, or using a surface function to determine an iso surface; see Sec. 5.5.4.

To implement this process on a GPU, all rays are stored in a queue and we process one ray at a time in a thread. The simulation domain intersection and traversal of the data structure is executed independently, on all threads in a warp, as these do not require any inter-thread communication. Next, after synchronizing the threads in a warp, we check if any thread has found a cell that can potentially contain fluid surfaces. For these potential intersections, we then calculate the actual ray-fluid intersection and store the results, if an intersection is found.

5.5.1 Data structure traversal

In case an intersection of a ray with the simulation domain AABB exists, this calculation yields a close distance λ^{min} and far distance λ^{max} to the domain intersections along the ray, where we clamp λ^{min} to positive values, which yields the distance along the ray, within the domain, as $\lambda_l = \lambda^{\text{max}} - \lambda_+^{\text{min}}$. Using λ_+^{min}, we determine $\mathbf{x}_{\text{min}} = \text{ray}(\lambda_+^{\text{min}})$, which in turn yields the integer coordinates of the first cell $\bar{\mathbf{x}}_{\text{min}}$ hit by the ray, within the simulation domain. Similar to Amanatides and Woo [AW87], we traverse the simulation domain by first calculating the integer ray direction as

$$\bar{\mathbf{d}} = [\text{sign}(\mathbf{d}^x), \text{sign}(\mathbf{d}^y), \text{sign}(\mathbf{d}^z)]\,. \tag{5.29}$$

We then determine the next cell boundary, intersected by the ray, as

$$\mathbf{x}_{\text{next}} = \mathbf{D}_{\text{min}} + \big(\bar{\mathbf{x}}_{\text{min}} + \max(\bar{\mathbf{d}}, \mathbf{0})\big) \circ \mathbf{C}_{\text{max}}, \tag{5.30}$$

where $\circ$ denotes the component-wise multiplication (Hadamard product) and the maximum is applied component-wise. Next, we determine the distance along the ray, until the next cell boundary is intersected, as

$$\lambda_n = \mathbf{x}_{\text{next}} - \mathbf{x}_{\text{min}} \oslash \mathbf{d}, \tag{5.31}$$

where $\oslash$ is the component-wise division (Hadamard division). For small components of $\mathbf{d}$, i.e. $|\mathbf{d}^x| < 10^{-6}$, we set the corresponding component of λ_n to infinite. Finally, we start the traversal in the cell $\bar{\mathbf{c}} = \bar{\mathbf{x}}_{\text{min}}$.

We first check if $\bar{\mathbf{c}}$ can contain fluid surfaces; see Sec. 5.4.3, and in this case stop the traversal, otherwise we move to the next cell. This increment, using $\bar{\mathbf{d}}$, is based on the smallest component in λ_n; however, if the smallest component λ_n was larger than λ_l, we stop the traversal. Otherwise we increment the smallest component in $\lambda_{\mathbf{n}}$, using $\bar{\mathbf{d}} \oslash \mathbf{d} \circ \mathbf{C}_{\text{max}}$, and repeat this process; see Algorithm 5.5.

ALGORITHM 5.5: Domain traversal algorithm; based on [AW87].

```
**Intersect** ray with simulation domain AABB, yielding $\lambda^{\text{min}}$ and $\lambda^{\text{max}}$
$\lambda_+^{\text{min}} \leftarrow \max(0, \lambda^{\text{min}})$
$\mathbf{x}_{\text{min}} \leftarrow \text{ray}(\lambda_+^{\text{min}})$
**Determine** $\bar{\mathbf{x}}_{\text{min}}$ using Eqn. 5.12
$\bar{\mathbf{d}} \leftarrow [\text{sign}(\mathbf{d}^x), \text{sign}(\mathbf{d}^y), \text{sign}(\mathbf{d}^z)]$
$\mathbf{x}_{\text{next}} \leftarrow \mathbf{D}_{\text{min}} + \left(\bar{\mathbf{x}}_{\text{min}} + \max(\bar{\mathbf{d}}, \mathbf{0})\right) \circ \mathbf{C}_{\text{max}}$
$\lambda_n \leftarrow \mathbf{x}_{\text{next}} - \mathbf{x}_{\text{min}} \oslash \mathbf{d}$
$\lambda_{inc} \leftarrow \bar{\mathbf{d}} \oslash \mathbf{d} \circ \mathbf{C}_{\text{max}}$
$\lambda_l \leftarrow \mathbf{x}_{\text{min}} - \text{ray}(\lambda_+^{\text{min}})$
**Initialize** $\bar{\mathbf{c}} = \bar{\mathbf{x}}_{\text{min}}$
**Iterate**
    **If** $\bar{\mathbf{c}}$ can contain fluid surfaces; see Sec. 5.4.3
        **Return** Cell found.
    **If** $\lambda_n^x < \lambda_n^y$
        **If** $\lambda_n^x < \lambda_n^z$
            **If** $\lambda_n^x > \lambda_l$: **Return** No cell found.
            $\bar{\mathbf{c}}^x \leftarrow \bar{\mathbf{c}}^x + \bar{\mathbf{d}}^x$        $\lambda_n^x \leftarrow \lambda_n^x \lambda_{\text{inc}}^x$
        **Else**
            **If** $\lambda_n^z > \lambda_l$: **Return** No cell found.
            $\bar{\mathbf{c}}^z \leftarrow \bar{\mathbf{c}}^z + \bar{\mathbf{d}}^z$        $\lambda_n^z \leftarrow \lambda_n^z + \lambda_{\text{inc}}^z$
    **Else**
        **If** $\lambda_n^y < \lambda_n^z$
            **If** $\lambda_n^y > \lambda_l$: **Return** No cell found.
            $\bar{\mathbf{c}}^y \leftarrow \bar{\mathbf{c}}^y + \bar{\mathbf{d}}^y$        $\lambda_n^y \leftarrow \lambda_n^y + \lambda_{\text{inc}}^y$
        **Else**
            **If** $\lambda_n^z > \lambda_l$: **Return** No cell found.
            $\bar{\mathbf{c}}^z \leftarrow \bar{\mathbf{c}}^z + \bar{\mathbf{d}}^z$        $\lambda_n^z \leftarrow \lambda_n^z + \lambda_{\text{inc}}^z$
```

5.5.2 Data structure visualization

For simulations with sparse data structures, visualizing the occupied cells is a useful, and fairly straight forward, visualization. To visualize the data structure, after the traversal algorithm yields a cell $\mathbf{c}$ potentially containing fluid surface, we check if there is a cell at location $\mathbf{c}$ in the simulation data structure, which is only the case for cells containing particles, not just fluid surface; see Sec. 5.4.1. If there is no cell, containing particles, we continue the traversal; otherwise, we calculate the center of the found cell as

$$\mathbf{x}_c = \mathbf{D}_{\text{min}} + \mathbf{c} \circ \mathbf{C}_{\text{max}}, \tag{5.32}$$

with a half cell size $r_c = 0.5\mathbf{C}_{\text{max}}$. Calculating the intersection of the ray with this cell yields λ_c^{min} and λ_c^{max} and the point of ray-cell intersection $\mathbf{x}_i = \text{ray}(\lambda_c^{\text{min}})$. Using $\mathbf{d} = \mathbf{x}_i - \mathbf{x}_c$, we determine a normal, where a cell is represented as a cuboid, as

$$\mathbf{n}_c = \left[(1+\epsilon)\,\mathbf{d} \oslash r_c\right], \tag{5.33}$$

where ϵ is a small positive value and $[\cdot]$ denotes rounding to the nearest integer. As the color for the intersected cell we use the color of the first particle in $\mathbf{c}$. This process yields all required information, i.e. depth, normal and color, required to render the data structure.

ALGORITHM 5.6: Intersection calculation after all threads in a warp have evaluated ϕ. *ballot*(p) is an intra-warp voting function of the predicate p, *shuffle*(t, i) returns the value of t for thread i, $active$ is a mask indicating active threads, and $ffs(b)$ returns the index of the first set bit in b. The result of this algorithm is a depth t.

```
In parallel for each thread $w$ in warp
    $\phi_w \leftarrow \phi(\mathbf{x}_w)$
    $\text{mask}_L \leftarrow ballot(\phi_s \geq \phi^{\text{iso}})$
    $\text{mask}_H \leftarrow ballot(\phi_s < \phi^{\text{iso}})$
    $\text{mask}_F \leftarrow (\text{mask}_L << 1)\, \& \text{mask}_H$
    $\text{mask}_R \leftarrow (\ \text{mask}_L << 1)\, \&\ \text{mask}_H$
    If $\text{mask}_L \& 0x1$
        $i \leftarrow \text{ffs}(\ \text{mask}_L) - 1$
    Else
        $i \leftarrow \text{ffs}(\text{mask}_L) - 1$
    If $\text{mask}_L \neq 0 \wedge \text{mask}_L \neq active$
        $x_0 \leftarrow i - 1$
        $y_0 \leftarrow \text{shuffle}(\phi_w, i - 1)$
        $x_1 \leftarrow i$
        $y_1 \leftarrow \text{shuffle}(\phi_w, i)$
        $dy \leftarrow = y_1 - y_0$
        $\alpha \leftarrow \phi^{\text{iso}} - y_0/dy$
        $t_0 \leftarrow \text{shuffle}(\lambda_w, i - 1)$
        $t_1 \leftarrow \text{shuffle}(\lambda_w, i)$
        $t \leftarrow (1 - \alpha)t_0 + \alpha t_1$
```

5.5.3 Particle visualization

Rendering particles as spheres, overlapping the cell $\mathbf{c}$ first involves finding the intersection of the current ray with the cell, similar to the cell visualization, which yields λ_c^{max} and λ_c^{min}. Using the simulation data structure, we iterate over all particles that can potentially be overlap $\mathbf{c}$, i.e. all those contained within neighboring cells of $\mathbf{c}$ and $\mathbf{c}$. For each of these particles j, we can calculate a ray sphere intersection, which, if there was an intersection, yields an intersection distance λ_j. If $\lambda_j \in [\lambda_c^{\text{min}}, \lambda_c^{\text{max}}]$, we store j and λ_j, if this was the closest intersection found so far, and keep iterating over the remaining particles. If there was an intersection with any particle, we can then determine the intersection point $\mathbf{x}_i = \text{ray}(\lambda_j)$ and the intersection normal $\mathbf{n}_i = \mathbf{x}_i - \mathbf{x}_j$. We use the color associated with the particle, i.e. using color mapping of some quantity, as the color for the intersection.

5.5.4 Surface rendering

Rendering the fluid surface directly requires extracting an iso surface of some field quantity ϕ. There are many different ways to determine this quantity; however, common choices include simply using the density ρ, the surface distance function of Zhu and Bridson [ZB05], or the anisotropic surface distance function of Yu and Turk [YT13]. For our approach the choice of function does not influence the process beyond potentially increasing the computational workload. Therefore, we assume an arbitrary field quantity $\phi(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$, with an iso value of ϕ^{iso}. To

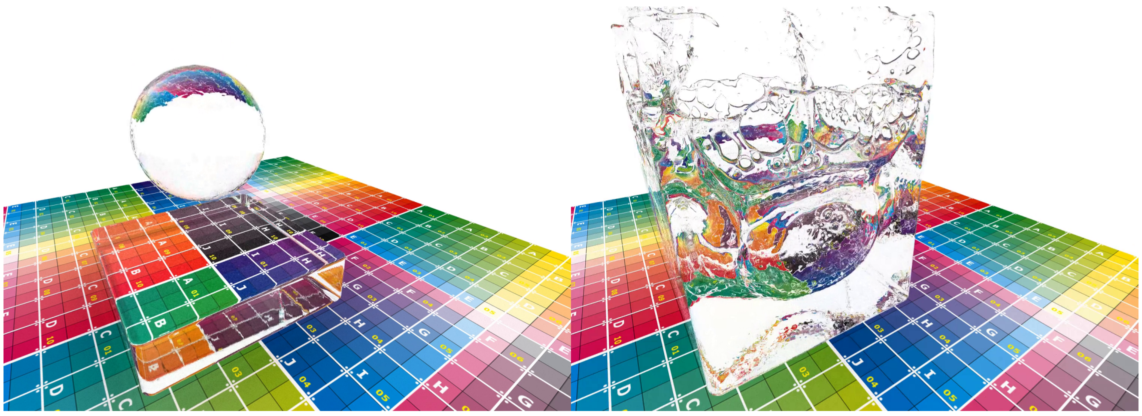


Figure 5.7: Rendering of the *drop* scene. A spherical fluid volume is dropped into a basin and rendered using an anisotropic surface function as a transparent fluid. The left part of the image shows the initial configuration, whereas the right part shows a later point in time.

calculate the ray-surface intersection we evaluate a single intersection of a ray with a corresponding found cell $\bar{\mathbf{c}}$ in parallel, using all threads in a warp, where we sequentially process the overall set of potential intersections.

To evaluate a single ray-surface intersection we first calculate the intersection of the ray with the found the cell $\bar{\mathbf{c}}$, yielding λ_c^{min} and λ_c^{max}. For a warp size of w, we then subdivide the interval $[\max\{\lambda_c^{\text{min}}, 0\}, \lambda_c^{\text{max}})$ into $w-1$ segments, where each segment $s \in [0, w-1)$ is assigned a starting position

$$\mathbf{x}_s = \text{ray}\left(\max\{\lambda_c^{\text{min}}, 0\} + \frac{s}{w-1}\left(\lambda_c^{\text{max}} - \max\{\lambda_c^{\text{min}}, 0\}\right)\right), \tag{5.34}$$

where each thread in a warp is assigned one position, with the last thread being assigned $\mathbf{x} = \text{ray}\,(\lambda_c^{\text{max}})$, and each ray is assigned an according distance value λ_s.

To evaluate ϕ_s, at each assigned position $\mathbf{x}_s$, we iterate over all particles in all neighboring cells, where the cell entries are stored in shared memory. Afterwards, utilizing Algorithm 5.6, we find an intersection depth λ_f and, if there was an intersection, store λ_f and mark the ray as processed; otherwise, we keep traversing the simulation domain for this ray. Calculating the intersection normal can be done directly by evaluating $\nabla\phi(\mathbf{x})$. Note the synchronization required here can be implemented implicitly by checking after each cell traversal if any thread in a warp has found a cell that can contain fluid surfaces and performing the check immediately. This avoids threads within a warp waiting for other threads and reduces warp divergence. We refer the reader to the supplementary materials for the implementation of this optimization, which can be found published along the definite version of the paper reprinted here; see [WK20].

5.6 Results and discussion

All results were simulated and rendered using a single Nvidia RTX 2080 Ti GPU with 11 GiB of VRAM and an AMD Ryzen 3970x CPU with 64 GiB of RAM. We used DFSPH [BK15], with a density error limit of 0.01% and a divergence error

Method	*Init*	Intersection-Tests				*Render*
		1st	*2nd*	*3rd*	*4th*	*Overall*
		Drop scene, Fig. 5.7				
[YT13]	12.1	90.1	113.4	103.5	84.8	838.7
		Dam break scene, Fig. 5.8				
[ZB05]	23	37.5	49.4	17.7	10.5	132.8
Particles	22.2	14.3	23.5	12.6	9.5	69.2
		Inlet scene Top Down, Fig. 5.9				
[ZB05]	12.7	30.4	41.8	19.1	14.6	136.7
[YT13]	14.1	149.8	204.2	60.6	46.9	503.6
Grid	12.7	2.3	3.4	1.7	1.3	27.86
Particles	12.6	12.9	19.9	13.7	10.9	79.8
		Inlet scene Side View, Fig. 5.10				
[ZB05]	12.8	46.9	52.7	25.3	18.2	180.2
[YT13]	14.0	237.5	273.3	124.7	79.5	785.3

Table 5.1: Timing for intersection tests. This table states the average computation time over the entire simulation in milliseconds per sample per pixel, separately for the first four intersection and the total rendering time including illumination calculations per sample per pixel. *Grid and* Particles stands for visualizing the underlying data structure and particles as spheres, respectively.

limit of 0.1%, and density maps [KB17] to represent rigid objects. Artificial viscosity was modeled based on XSPH [Mon05], surface tension was modeled based on [AAT13], and fluid air phase interactions were modeled based on [Gis+17]. We used the vorticity refinement method of [Ben+18] and the spatially adaptive method of [WHK17]. Our overall uni-directional ray-tracing algorithm is implemented in CUDA and inspired by [PJH16], where we used a simple bounding volume hierarchy for rigid objects, and calculate each bounce of a ray using a loop on the CPU to avoid recursions on the GPU. We implemented our data structure and rendering approach in the open source SPH framework openMaelstrom [Win19]. For an in-depth discussion of the simulation data structure we refer the reader to [WK19]. For all renderings we used a fixed resolution of $1920x1080$ and a framerate of 60 fps, with 35 primary rays cast per pixel with 6 and 16 bounces per primary ray for opaque and transparent renderings, respectively. .

Test Scenes: We evaluated our approach using three test scenes. The *Dam break* test scene involves the collision of two initial fluid volumes in a cubic domain; see Fig. 5.8, with a particle resolution of $r = 0.175m$ and 3.9 million particles. The *Inlet* test scene involves a stream of liquid flowing along a channel and colliding with an obstacle; see Fig. 5.10, with a particle resolution of $r = 0.35m$ and up to 2.2 million particles. The *Drop* test scene involves dropping a sphere of liquid into a basin; see Fig. 5.7, with a particle resolution of $r = 0.2m$ and 1.5 million particles. For simulation performance see Table 5.2, and for rendering performance see Table 5.1.

Scene	Fig.	Particles	Radius/m	Δt/ms	Domain	Structure/ms	Neighbor-list/ms	DFSPH/ms	Overall/ms
Drop	5.7	1.5M	0.2	4.0	$68x68x204$	30.5	77.3	1150.6	2079
Dam break	5.8	3.9M	0.175	5.6	156^3	41.5	174.9	2312	2919
Inlet	5.9	2.2M	0.35	6.0	$156x156x76$	22.1	66.8	316.2	517

Table 5.2: The timing values shown here are given as the average time, required to simulate 1/60s, over an entire simulation as measurements in milliseconds. DFSPH refers to the combined time required to solve for both incompressibility and divergence-freedom. The timing for the data structure here refers to all steps in Algorithm 5.2, i.e.,including resorting and data structure creation. Neighbor-list timings refer to the timing of the construction of a constrained neighbor-list [WK19].

Ray-casting performance: To evaluate the performance of our method, we first consider the performance cost of the primary ray intersection. These rays are cast directly from the camera into the scene and traverse similar cells. Primary ray intersections can be used to implement a deferred shading approach, as the intersection yields both a depth and normal, or using a simple direct lighting model. When using an isotropic surface function [ZB05], we find good performance in all scenes, i.e. primary intersection tests requiring less than $50ms$. For an anisotropic surface function [YT13] the overall computational cost is significantly higher, due to a more expensive surface function and larger cell non-cubic cells. The cost of the primary intersection, regardless of which surfacing method specific is chosen, changes significantly depending on the relation of camera and fluid geometry. This involves many aspects, i.e. many rays parallel to a flat surface require many unnecessary intersection tests, whereas rays orthogonal to a flat surface require few unnecessary intersection tests. Additionally the performance changes depending on how many rays can even intersect the fluid, i.e. the fluid surface might be partially occluded by some other geometry, and also depends on the screen-space size of the fluid simulation. Finally, the overhead for the construction of the render data structure is fairly low and is only required once per frame.

Opaque fluid rendering: Rendering an opaque surface does not just require the primary intersection, but a full path trace per ray. The directions of bounced rays show very little spatial coherency, which causes the rays to not traverse similar cells, as was the case for the primary intersection. In all scenes we tested (see Tab. 2), we found that the performance of the first bounced ray is always lower than the primary ray, and the cost of following bounces decreases with each bounce. However, if the number of rays intersecting the fluid domain falls below the number of threads we can executed in parallel, i.e. after a significant number of bounces, the cost stays relatively fixed. Additionally, the performance depends on the geometry of the surface, i.e. any bounced ray from an intersection with a cube shared surface is guaranteed to not intersect the surface directly. However, more complex geometries that, for instance, contain cavities where rays enter but rarely exit, significantly increase computational cost. Furthermore, for an opaque rendering we require multiple samples per pixel, which further increases the computational cost.

Transparent fluid rendering: Whereas an opaque surface rendering does not cause rays to transmit into the fluid, a transparent surface rendering will cause rays to either reflect or refract into the fluid. This means that the computational requirements are significantly higher, as rays traversing the interior of the fluid require many checks for potential fluid intersections, which are mostly negative as a significant number of internal cells could contain fluid surface. Additionally, rays moving on the interior of a fluid volume might be reflected multiple times before they exit the fluid, requiring significantly more bounces per ray than an opaque surface rendering. These additional computational costs increase the time per sample-per-pixel by about 3 times. Even though there are options to improve the computational performance, such as lowering the number of samples per pixels, utilizing a screen-space based refraction model [LGS09], or considering refraction on the first bounce only, these methods have a large and complex impact on visual quality, the study of which is beyond the scope of this article. Even though our method is slowed down for a transparent rendering, we can still achieve good

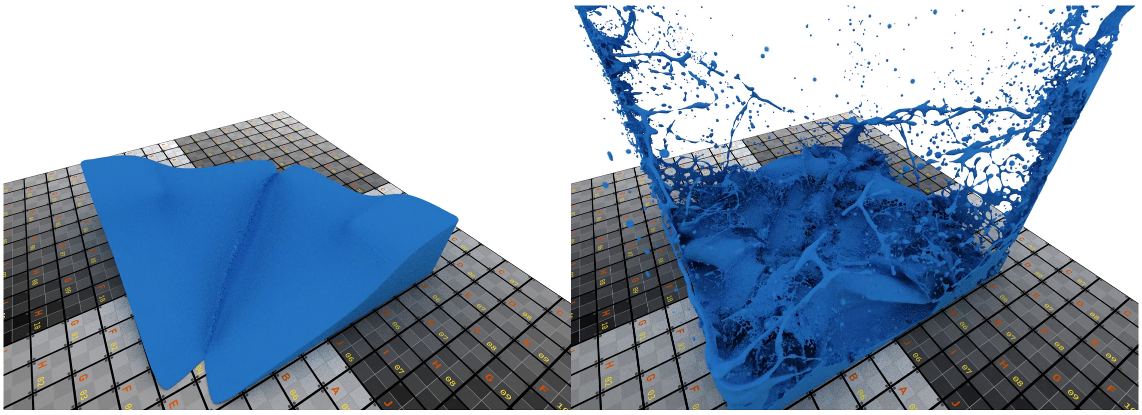


Figure 5.8: Rendering of the *dam break* scene. Two initial fluid volumes collide in this scene and are rendered using an isotropic surface function as an opaque fluid. The left part of the image shows the simulation immediately after the collision, whereas the right part shows a much later point in time.

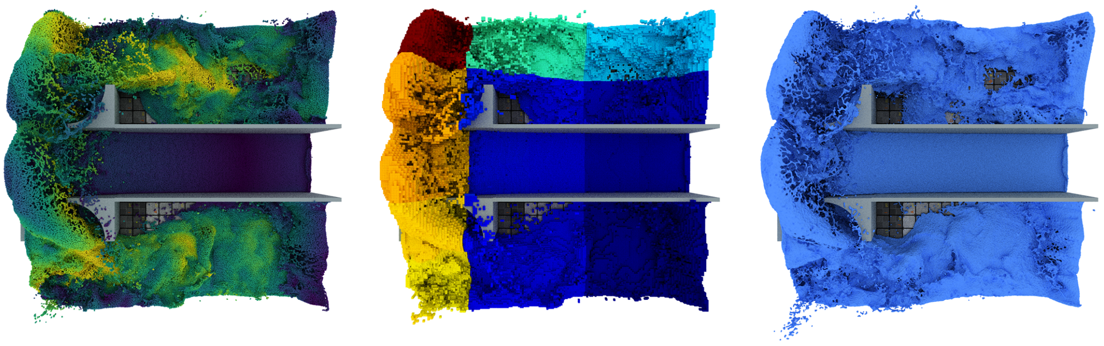


Figure 5.9: Rendering of the *inlet* scene. A fluid inlet stream impacts a rigid obstacle visualized as particle spheres, with particle velocities color coded from purple ($0m/s$) to yellow ($30m/s$) (left), a visualization of the data structure, with the Morton code of the first particle in each cell color coded from blue to red (middle), and an isotropic surface extraction (right).

results, especially using anisotropic surface functions, just not at interactive framerates; see Tab. 2.

Simulation data structure memory usage: Our data structure requires a hash map and a cell list, both containing a begin and length entry; see Sec. 5.4.4. The length of the cell list is determined by the number of particles, i.e. $\mathcal{O}(n_{\text{particles}})$, whereas the hash table size is determined as the smallest prime number larger than the number of particles. As this value is very close to the number of particles, we can assume that the hash map scales with the number of particles. Using bitfields we require 4 Bytes per hash map entry and 8 Bytes per cell table entry. In contrast to many prior methods, the memory requirements only depend on the particle count, and not the simulation domain or particle distribution.

Rendering data structure memory usage: Our rendering data structure is similar to the simulation data structure and, as such, also requires 12 Bytes per particle, for a 32 Bit Morton Code. Overall, as the simulation data structure is required for rendering, the total cost per particle is 24 Bytes. However, ignoring memory requirements for the overall rendering algorithm, i.e. for textures, these

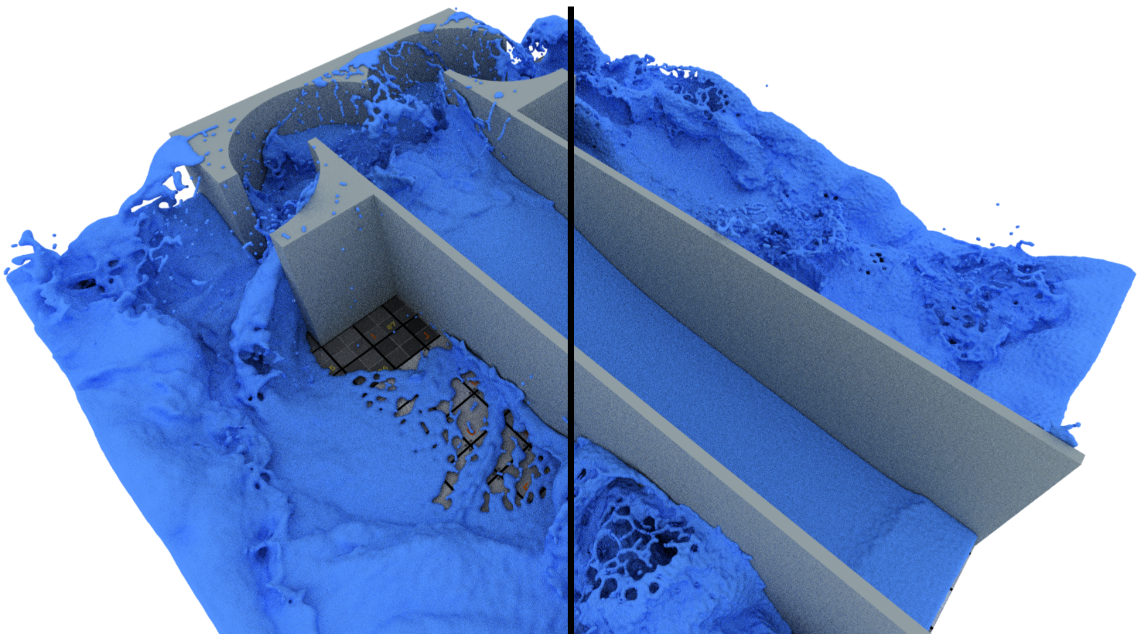


Figure 5.10: Rendering of the *inlet* scene using an anisotropic (left) and an isotropic surface function (right).

memory requirements only scale with the particle count, i.e. $\mathcal{O}(n_{\text{particles}})$, and are independent of domain size, particle distribution or particle resolution, similar to the simulation data structure. Furthermore, we do not require any amount of memory to store an extracted mesh, nor an acceleration structure for the mesh, nor an intermediate grid used for an explicit surface extraction.

Limitations: Even though our method yields very high quality surfaces at a low, and constant, memory cost, the computational requirements for the surface extraction can become very high, especially when using complex surface functions, i.e. [YT13]. This performance cost becomes especially noticeable in transparent surface renderings, making them comparably slow. While our approach is ready to render adaptive fluid surfaces [WHK17], the performance quickly degrades with higher adaptivity ratios, as applying the same approach as was done for the simulation (see Sec. 5.4.2) is not directly possible.

5.7 Conclusions

In this paper, we presented a data structure that can be used to efficiently simulate and render SPH fluid simulations using a combination of a cell table and hash map. The memory requirements of our approach do not scale with the domain size, particle resolution, or particle distribution, but instead solely depend on the number of particles, making our approach well suited for usage on GPUs. Using our simulation data structure [WK19], we can efficiently simulate even highly adaptive SPH simulations in unbounded simulation domains. Using our rendering data structure, we can efficiently render an iso surface, without requiring an explicit surface extraction, using arbitrary surface metrics, i.e. isotropic and anisotropic, using opaque or transparent shading models.

Chapter 6

Semi-Analytic Boundary Handling Below Particle Resolution for Smoothed Particle Hydrodynamics

Contextualization

This chapter reprints the paper "Semi-Analytic Boundary Handling Below Particle Resolution for Smoothed Particle Hydrodynamics" published as proceedings of ACM SIGGRAPH Asia 2020 via ACM Transactions on Graphics [WAK20]. In this paper, a boundary model is proposed that is both consistent across varying resolution scales, i.e., evaluations are consistent for particles before and after changing resolution, and that does not require particle samplings of boundary geometries. Overall, this method plays an important role in adaptive SPH simulations as without a consistent boundary model interactions with boundary geometries have to occur with fluid particles of consistent resolution, which, in turn, limits the applicability of adaptive methods.

The core idea of this paper is that for adaptive simulations the resolution of a fluid particle could be chosen arbitrarily small, i.e., to a point where boundary geometries appear as locally planar and particles of any resolution should converge towards this behavior with decreasing resolution. Using this assumption and an analytic solution for this specific geometric case, which only depends on the particle-boundary distance, the boundary geometry does not change with changing particle resolutions and, consequently, remains water-tight and numerically stable across varying solutions. Furthermore, this assumption reduces the smoothing of boundary features that is inherent to SPH, especially at low fluid resolutions, yielding much sharper and consistent boundary geometries.

The main idea of using locally planar boundary geometries came from Rene Winchenbach, who also provided the necessary derivation of the analytic integral terms. Rustam Akhunov provided the implementation of the particle based boundary handling and provided feedback on the argumentation of the paper during many discussions in our shared office. Andreas Kolb helped in the writing process of the paper and provided valuable help regarding the argumentation concerning signed distance fields. Thanks is also owed to the anonymous reviewers, who initially rejected the paper for SCA 2020, as they provided many valuable insights that helped significantly in re-structuring the paper into its final published form.

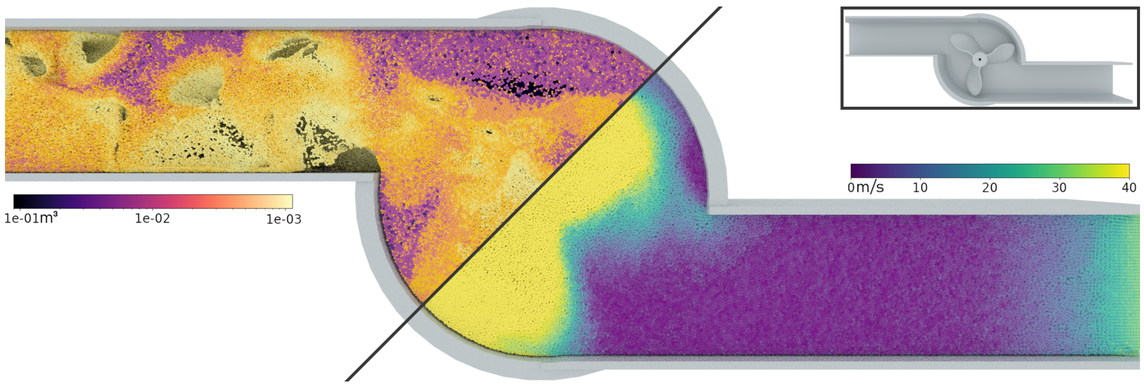


Figure 6.1: Our novel semi-analytical boundary handling method enables fluid-rigid interactions even under difficult conditions and can be directly combined with any spatially adaptive simulation technique. This figure shows the simulation of a collision of an inlet flow from the right with a counterclockwise rotating propeller, calculated with up to 1.8 million particles and an adaptive volume ratio of $100:1$. The boundary configuration is shown in the top right corner, the left and the right part of the main figure visualizes particle volume and particle velocity, respectively.

Abstract

In this paper, we present a novel semi-analytical boundary handling method for spatially adaptive and divergence-free smoothed particle hydrodynamics (SPH) simulations, including two-way coupling. Our method is consistent under varying particle resolutions and allows for the treatment of boundary features below the particle resolution. We achieve this by first introducing an analytic solution to the interaction of SPH particles with planar boundaries, in 2D and 3D, which we extend to arbitrary boundary geometries using signed distance fields (SDF) to construct locally planar boundaries. Using this boundary-integral-based approach, we can directly evaluate boundary contributions, for any quantity, allowing an easy integration into state of the art simulation methods. Overall, our method improves interactions with small boundary features, readily handles spatially adaptive fluids, preserves particle-boundary interactions across varying resolutions, can directly be implemented in existing SPH methods, and, for non-adaptive simulations, provides a reduction in memory consumption as well as an up to $2\times$ speedup relative to current particle-based boundary handling approaches.

6.1 Introduction

In modern computer animation the physically accurate simulation of high quality free surface liquid systems is becoming ever more important, but uniform resolution increases are strongly limited by available computational resources. However, not all areas of a simulation are equally important for the outcome, i.e., for computer animation the surface detail is more important than the fluid's bulk, and as such methods have been developed that focus the computational resources in regions of interest, which allow for far greater surface detail than would be possible with a uniform resolution at the same computational requirements. Recent work has enabled incompressible spatially adaptive simulations with large refinement ratios for Smoothed Particle Hydrodynamics (SPH) [WHK17]. So far, the robust handling of arbitrary boundaries is an unsolved challenge in adaptive SPH, i.e.,methods are required that provide consistent interactions with arbitrary boundaries across widely varying resolution scales. Additionally, representing small boundary features requires significantly higher fluid resolutions, relative to boundary feature sizes.

In SPH boundaries are commonly described using a particle-based representation [Aki+12], which places boundary particles on the surface of an object, or using some direct representation of the boundary [Ben+19], i.e., by pre-calculating boundary integrals based on, for instance, grid-based integration schemes. Particle-based methods suffer from a dependence on the sampling quality and while there are methods that can change the particle distribution on-the-fly, there are no methods that can change the resolution of a boundary sampling on-the-fly. Direct boundary representation methods suffer from a dependence on the sampling resolution. Koschier and Bender [KB17] precompute a boundary-integral on a regular grid, which is only correct for one resolution, whilst Fujisawa and Miura [FM15] utilize empirical methods, to enhance a semi-analytical solution, which are not transferrable across varying particle resolutions. In Computational Fluid Dynamics (CFD) boundaries are commonly described using wall-renormalization approaches; see Feldman and Bonet [FB07]. These can be realized in many different ways, i.e., using semi-analytical methods [Chi+19]. None of these methods can be directly applied to adaptive SPH as they depend on some sampling of the boundary that needs to be adapted to the particle resolution. Moreover, none of the existing methods can handle small boundary features at or below particle resolution properly, i.e., integral-based formulations smooth them out and particle-based methods yield uneven boundaries.

In our paper, we propose a novel-semi analytic boundary handling approach that comprises the following major contributions:

- An analytic, scale independent, solution to the interaction of particles with a planar boundary.
- A signed distance field based approach that handles the interaction of a particle with an arbitrary boundary by local planar approximations.
- Significantly improved handling of small boundary geometries at, or even below, the current particle resolution.
- A boundary interaction scheme based on a single contact that allows for easy pressure estimates and two-way coupling.

Our boundary handling scheme can be easily integrated into existing simulation methods and other boundary effects, e.g., friction, can be adapted without conceptual modifications. We evaluate our method in a variety of simulations to demonstrate its benefits compared to prior approaches, especially considering boundary representation.

6.2 Related work

In the following we will mostly focus on boundary handling methods in SPH and refer the reader to Koschier et al. [Kos+19] for a wider overview of recent work in SPH in computer animation contexts.

Incompressibility plays an important role in realistic fluid simulations. Whereas initial work involved weakly compressible (WCSPH) methods [MCG03] and the predictive-corrective incompressible (PCISPH) method [SP09], recent work focuses on implicit formulations (IISPH) [Ihm+13]. Following IISPH, Bender and Koschier [BK15] proposed the current state of the art approach of enforcing incompressibility by adding an additional divergence-free solver. Many recent advances, considering incompressibility and stability, have been achieved by incorporating the boundary representation more directly into the SPH model, i.e., by pressure extrapolation [Ban+18b], or by using particles for rigid-rigid interactions for strong rigid coupling (ILSPH) [Gis+19]. Macklin and Müller [MM13] developed a different SPH simulation method utilizing position-based dynamics (PBD), which was later integrated into a unified simulation model for fluid and rigid systems [Mac+14].Moreover, spatially adaptive methods have been widely researched in the past, for example by Adams et al. [Ada+07] for WCSPH, by Solenthaler and Gross [SG11] using non mass-preserving approaches and recently by Winchenbach et al. [WHK17] for incompressible SPH methods. However, these methods usually assume that a (nearly) fixed particle resolution is in contact with the boundary.

Boundary handling in SPH has been a longstanding research topic and various different approaches have been proposed to handle fluid-boundary interactions in the last decade. In general, there exist four categories of boundary handling approaches:

1. External boundary handling methods
2. Particle-based methods
3. Wall-renormalization methods
4. Boundary-integral methods

External boundary handling methods were initially developed for weakly compressible SPH (WCSPH) utilizing a variety of approaches, i.e., particle level sets [Los+08] or direct forcing [BTT09], however these methods are limited in their applicability as they cannot be tightly integrated into SPH methods.

Particle-based methods either handle boundary interactions using ghost particles representing boundary objects [AHA12], or using explicitly sampled boundary particles placed on the surface of a boundary object[Aki+12]. However, particle-based methods suffer from a strong dependence on the quality of the sampling. Band et al. [BGT17] proposed a particle-based boundary representation with improved accuracy in flat regions, by using local planar boundary fitting and resampling of the particle representation to reduce the sampling problems inherent to particle-based representations. Band et al. [Ban+18a] proposed an extended boundary handling scheme that directly estimates pressure values on boundary particles, which has been further optimized by a moving-least-squares-based pressure extrapolation [Ban+18b].

Wall-renormalization approaches extend the fluid domain into the boundary domain and were initially proposed by Feldman and Bonet [FB07]. Ferrand et al. [Fer+13] utilized this approach to realize a semi-analytical boundary handling scheme in 2D, which was later extended to 3D by Maryhofer et al. [May+15]. Leroy et al. [Ler+14] further extended this approach to handle incompressible SPH methods. However, these methods often only apply to 2D or viscous flows and can typically not be applied directly to adaptive simulations [Chi+19]. Furthermore, these approaches cannot simply be integrated into computer animation approaches directly, as these approaches require an additional renormalization term on top of the direct summation of contributions.

A semi-analytical *boundary-integral method* was proposed by Fujisawa and Kenjiro [FM15], that utilized empirically derived functions to allow for incompressible SPH simulations in simple one-way coupled scenarios. Recently, Koschier and Bender [KB17] developed a Boundary-integral method that pre-calculates boundary integrals on a fixed numerical grid, using an expensive pre-calculation method. This was improved by Bender et al. [Ben+19] by pre-calculating a modified volume term. However, these methods either require expensive pre-computations or rely on empirically derived functions.

6.3 Foundations Of boundary handling in SPH

After a short introduction to the fundamental aspects of SPH, this section gives a brief conceptual overview of the various boundary handling approaches, and how they affect SPH. This serves as a basis for the derivation of our method in the upcoming sections.

The underlying basis of SPH is an approximation of an integral identity using smoothing kernels with compact support radii. The integral identity of a function $A : \Omega \rightarrow \mathbb{R}^m$, defined over a domain $\Omega \subset \mathbb{R}^n$, can be written, using the Dirac delta function δ [GM77], as

(6.1) $$A(\mathbf{x}) = \int_\Omega A(\mathbf{x}')\delta(\mathbf{x}-\mathbf{x}')d\mathbf{x}'.$$

Replacing the Dirac delta function with a radially symmetric smoothing kernel $W : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$, defined over a spherical domain $\mathcal{D}_\mathbf{x}$, around $\mathbf{x}$ with radius h, commonly referred to as support radius, yields an approximation for A as

(6.2) $$A(\mathbf{x}) \approx \int_{\mathcal{D}_\mathbf{x}} A(\mathbf{x}')W(|\mathbf{x}-\mathbf{x}'|, h)d\mathbf{x}'.$$

It is important to note that this approximation will only yield exact results, i.e., exactly representing a boundary geometry, if the support radius approaches 0, and thus (6.2) yields (6.1) again. For all nonzero support radii, this approximation introduces a smoothing of any quantity over a spatial domain, limiting the achievable resolution of an SPH simulation for a given support radius. The smoothing kernel can be defined as [DA12]

(6.3) $$W(r,h) = \frac{1}{h^n}C_n\hat{W}\left(\frac{r}{h}\right),$$

where n denotes spatial dimensionality, C_n is a normalization factor such that $\int_{\mathcal{D}_{\mathbf{x}}} W(|\mathbf{x}' - \mathbf{x}|, h) d\mathbf{x}' = 1$ and $\hat{W}$ being the actual kernel. Various kernel functions exist, however, we will focus on the cubic spline kernel, as it is widely used in computer animation. The cubic spline kernel function has a normalization factor of $\frac{80}{7\pi}$ in 2D and $\frac{16}{\pi}$ in 3D with the kernel defined as [DA12]

(6.4) $$\hat{W}_{\text{spline}}(q) = [1-q]_+^3 - 4\,[0.5-q]_+^3\,,$$

with $[\cdot]_+ = \max(0,\cdot)$. We define $\mathcal{D}_{\mathbf{x}}$ as the local boundary domain as the overlap of the overall boundary domain Ω_B with the support domain, $\mathcal{D}_{\mathbf{x}}^B = \mathcal{D}_{\mathbf{x}} \cap \Omega_B$, as well as the local fluid domain, $\mathcal{D}_{\mathbf{x}}^F = \mathcal{D}_{\mathbf{x}} \setminus \mathcal{D}_{\mathbf{x}}^B$. If there is no local boundary domain, i.e., $\mathcal{D}_{\mathbf{x}}^B = \emptyset$, (6.2) can directly be discretized using fluid particles only, where each particle carries a mass m and has a density ρ, yielding [Mon92]

(6.5) $$A(\mathbf{x}) \approx \sum_j A_j \frac{m_j}{\rho_j} W(|\mathbf{x}_j - \mathbf{x}|, h),$$

where j is the set of neighboring fluid particles within h, with respect to $\mathbf{x}$. For further details on this derivation, as well as gradient terms, we refer the reader to [Pri12] and [Kos+19].

However, this discretization is only applicable in the absence of any overlap with a boundary domain. The different categories of boundary handling methods (see Sec. 6.2) treat regions containing some boundary segment significantly differently.

External boundary handling methods do not discretize, or evaluate, (6.2) for boundary regions, i.e., they use (6.5) everywhere and enforce boundary handling through some external mechanism, e.g., by clamping particle positions. These methods are not commonly used in modern simulation methods as they cannot be tightly integrated into an SPH method like IISPH or DFSPH.

Particle-based methods involve sampling the surface of a boundary domain with particles [Aki+12]. Here each boundary particle b has a rest volume $V_b^0 = \frac{\gamma}{\sum_{b_b} W_{bb_b}}$ [Ban+18a], depending on neighboring boundary particles b_b. This γ factor is used to correct for the lack of particles sampling the interior of the boundary and is commonly set to 0.7 to prevent penetration [Gis+19]. This yields a modified SPH estimate including boundary particles as

(6.6) $$A(\mathbf{x}) \approx \sum_j A_j \frac{m_j}{\rho_j} W_{ij} + \sum_b A_b \frac{V_b^0}{\delta_b} Wib,$$

with $\delta_b = \sum_{b_f} V_{b_f} W_{bb_f} + \sum_{b_b} V_{b_b}^0 W_{bb_b} + \beta$. This formulation allows a natural extension of many SPH processes to boundary interactions and also enables strong rigid-fluid coupling [Gis+19]. However, these interactions strongly depend on the resolution and quality of the sampling and are not applicable to spatially adaptive methods. Furthermore, sampling all regions of a boundary domain, even if they only briefly come into contact with fluid particles, can require significant amounts of extraneous boundary particles.

Wall-renormalization methods, commonly found in CFD contexts, determine a wall-renormalization term γ [Ler+14] that describes the relation between $\mathcal{D}_{\mathbf{x}}$ and $\mathcal{D}_{\mathbf{x}}^F$, as

(6.7) $$\gamma(\mathbf{x}) = \int_{\mathcal{D}_{\mathbf{x}}^F} W(|\mathbf{x} - \mathbf{x}'|, h) d\mathbf{x}',$$

This term is commonly evaluated using the divergence theorem, i.e., by evaluating (6.7) over the surface of the local boundary domain $\partial\mathcal{D}^B_{\mathbf{x}}$ using semi-analytical approaches; see [Chi+19]. This yields a modified SPH estimate as

$$A(\mathbf{x}) \approx \frac{1}{\gamma(\mathbf{x})}\sum_j A_j \frac{m_j}{\rho_j} W_{ij}. \tag{6.8}$$

The evaluation of wall-renormalization methods, i.e., due to the γ term, is computationally expensive and assumes that the fluid domain can be extended into the boundary domain. The latter is, however, not always correct, for instance in case of a moving boundary object with a velocity independent of the fluid.

Boundary-integral methods directly evaluate the integral (6.2) over the boundary domain [KB17], i.e.,

$$A(\mathbf{x}) \approx \sum_j A_j \frac{m_j}{\rho_j} W_{ij} + A_b \int_{\mathcal{D}^B_{\mathbf{x}}} W(|\mathbf{x}_i - \mathbf{x}'|, h) d\mathbf{x}', \tag{6.9}$$

with A_b representing the relevant quantity for the boundary domain. The boundary contribution $\lambda(\mathbf{x}) = \int_{\mathcal{D}^B_{\mathbf{x}}} W(|\mathbf{x}-\mathbf{x}'|, h) d\mathbf{x}'$ is either pre-calculated [KB17] or approximated [FM15]. Note that, as (6.2) is equal to 1 over $\mathcal{D}_{\mathbf{x}}$, the wall-renormalization factor $\gamma(\mathbf{x})$ is $1-\lambda(\mathbf{x})$. Precomputed values for $\lambda(\mathbf{x})$ are only valid for a specific particle resolution, as the value depends on the support radius, and empirically-based approaches are not consistent across varying particle resolutions.

6.4 Method overview

Our novel semi-analytic boundary-integral-based method is built on the idea of treating all boundaries as locally planar and utilizing an analytic solution for $\lambda(\mathbf{x})$ to evaluate the contributions of all boundaries. This requires us to find a locally planar boundary, representing an overall boundary domain, for each boundary domain that a single particle interacts with. This means that (6.9) becomes

$$A(\mathbf{x}) \approx \sum_j A_j \frac{m_j}{\rho_j} W_{ij} + \sum_b A_b \lambda^b_{i,n}, \tag{6.10}$$

where b are the boundary domains intersecting $\mathcal{D}_{\mathbf{x}}$ and $\lambda^b_{i,n}$ describes the interaction of particle i with the planar boundary representing b in n dimensions for the position $\mathbf{x}_i$.

The assumption of local flatness, as shown in Fig. 6.2, works well for smooth boundaries where the boundary surface has a well defined first derivative at all positions, which can readily be utilized to find a local plane that becomes a better approximation as the support radius shrinks. Spatially adaptive methods can be used to ensure that high resolution particles are placed in areas of high boundary curvature to improve this approximation further.

Conceptually, directly evaluating (6.2) leads to a smoothing effect for sharp features, i.e., features that are as large as, or smaller than, the support radius are not properly represented. This smoothing causes the boundary shape to not be properly represented, e.g., sharp corners become smoothed out, which results in a strong dependence of boundary representation on particle resolution. Our

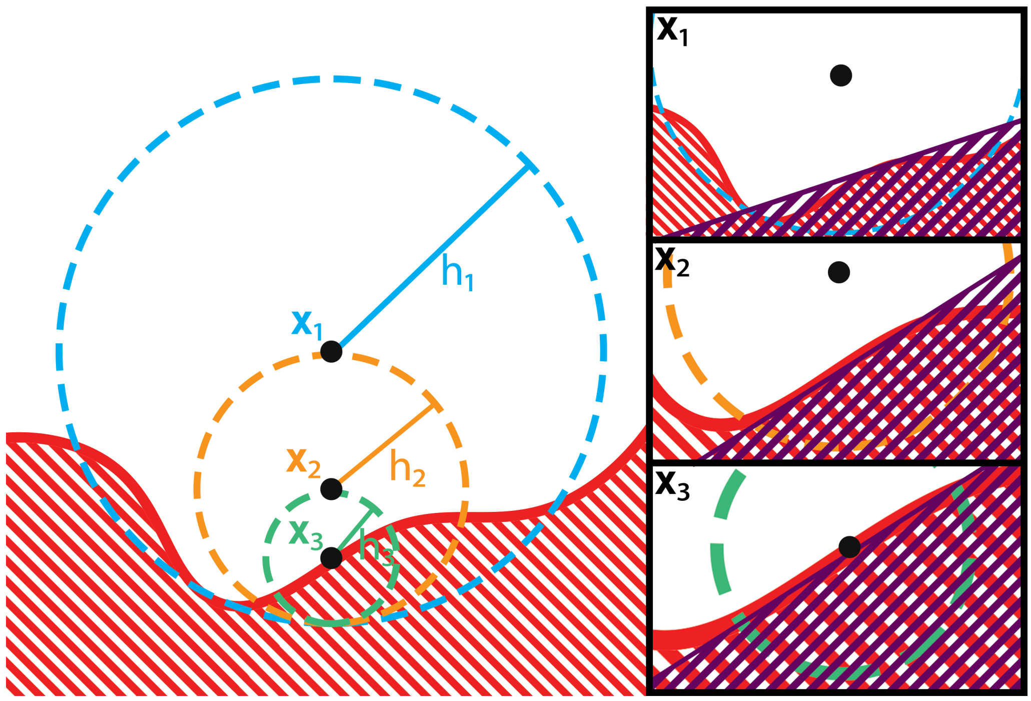


Figure 6.2: Considering the boundary as locally flat is less accurate, i.e., for position $\mathbf{x}_1$ and support radius h_1(top right). However, with smaller support radii the assumption leads to much better representations (bottom right).

method avoids this by ensuring that the locally planar representation of a boundary coincides with the actual boundary geometry such that $\int_{\mathcal{D}^B_{\mathbf{x}}} W(|\mathbf{x}-\mathbf{x}'|, h)d\mathbf{x}' = 0.5$, on the boundary surface. This means that particles, regardless of the geometry of the boundary domain, or the particle resolution, will be influenced solely based on the distance to the boundary surface, which in turn means that boundary features are not smoothed out. We will evaluate the differences in boundary-representation and how they influence the simulation, especially with regard to small features, in Sec. 6.9.2.

6.5 Analytic planar boundaries

To find an analytic solution for planar boundaries, we initially assume that particles have a support radius of 1 and demonstrate the analytic solutions in 2D and 3D; see Sec. 6.5.1 and 6.5.2 respectively. Sec. 6.5.3 extends the solution to arbitrary support radii. Sec. 6.5.4 discusses gradients and related terms. Sec. 6.5.5 discusses penalty terms.

6.5.1 2D analytical boundary integrals

The boundary contribution term in 2D, for a planar boundary, only depends on the distance to the plane, which means that

$$\lambda_2(d) = \int_0^1 \int_0^{s(q)} C_2 \hat{W}(q) dl dq, \tag{6.11}$$

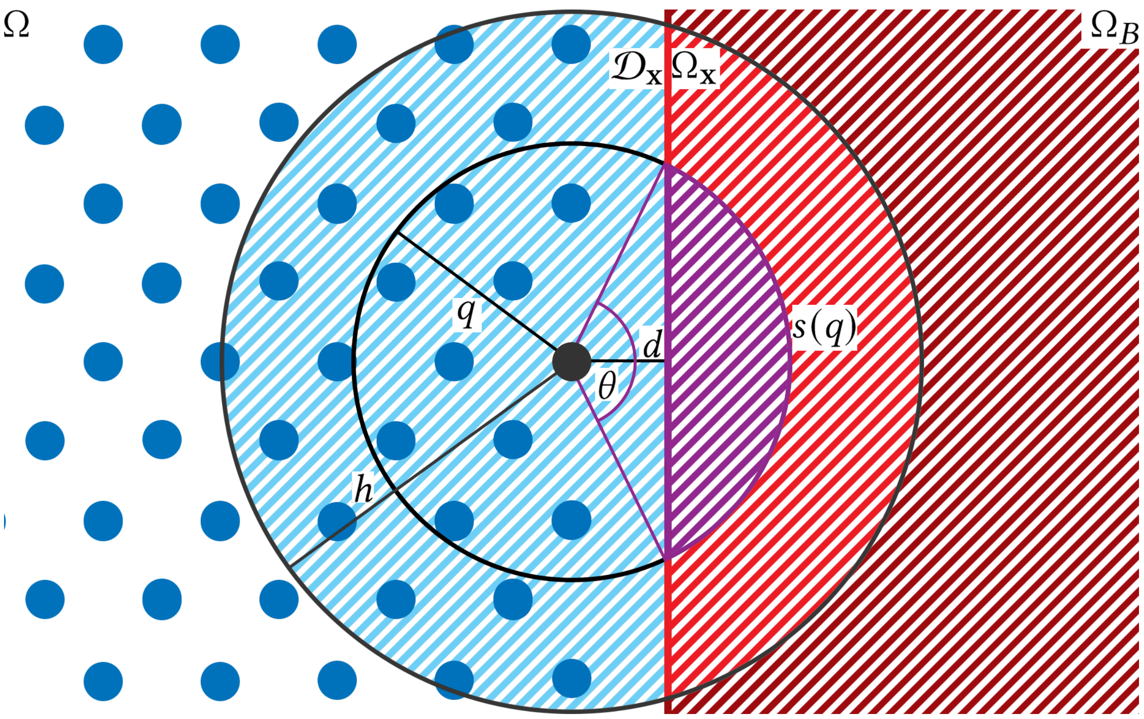


Figure 6.3: This figure shows a planar boundary in 2D. Here a particle contains a fluid region, $\mathcal{D}^F_\mathbf{x}$, and a boundary region, $\mathcal{D}^B_\mathbf{x}$, within $\mathcal{D}_\mathbf{x}$. For a given relative distance q and a boundary distance d, we find an angle θ, which describes an arc segment $s(q)$, which we use to determine the integral.

for spherical coordinates q and l, where $s(q)$ is the length of an arc with radius q, centered d away from the boundary, that is within $\mathcal{D}^B_\mathbf{x}$, i.e., $s(q) = 2q\pi$ for $\mathcal{D}^B_\mathbf{x} = \mathcal{D}_\mathbf{x}$. As we assume $\mathcal{D}^B_\mathbf{x}$ to be locally flat, $s(q)$ is the arc length of a circle segment C, based on the signed distance d from $\mathbf{x}$ to $\mathcal{D}^B_\mathbf{x}$; see Fig. 6.3. Using standard algebra we get

(6.12)
$$s(q) = 2q \operatorname{acos}\left(\frac{d}{q}\right),$$

for $d > 0$. If $q < d$ the arc length $s(q)$ would result in a complex number and as such we exclude this region by increasing the lower bound for q to be at least d, for now. Inserting this into (6.11) yields

(6.13)
$$\int_d^1 \int_0^{2q\operatorname{acos}\left(\frac{d}{q}\right)} C_2\hat{W}(q)dl dq = \int_d^1 C_2\hat{W}(q)2q\operatorname{acos}\left(\frac{d}{q}\right)dq.$$

We can readily extend this integral for $d < 0$ using the rotational symmetry of the smoothing kernel as

(6.14)
$$\lambda_2(d) = \int_0^1 C_2\hat{W}(q)2\pi q^2 dq - \lambda_2(-d) = 1 - \lambda_2(-d).$$

In order to evaluate the integral in (6.13) we have to specify the kernel being used. Therefore, we employ the cubic spline kernel function (6.4), although any other integrable smoothing kernel can be used. As the cubic spline kernel is defined piecewise, we need to split the integral for $d \leq 0.5$ into two definite integrals over $[d, 0.5]$ and $(0.5, 1]$, whereas for $d > 0.5$ we only need to consider the definite integral over $[d, 1.0]$. These definite integrals, shown in the appendix as (6.56), can be solved symbolically, and the final expression for $\lambda_2(d)$ is given in the Appendix in (6.57).

6.5.2 3D analytical boundary integrals

In 3D, we analogously rewrite the boundary contribution term as an integral over the 3D unit ball V, using spherical coordinates, as

$$\lambda_3(d) = \iiint_V C_3 \hat{W}(q) dV = \int_0^1 \iint_0^{A(q)} C_3 \hat{W}(q) dA dq, \tag{6.15}$$

where $A(q)$ denotes the partial area of a sphere of radius q inside the boundary domain $\mathcal{D}_{\mathbf{x}}^B$, i.e., $A(q) = 4\pi q^2$ for $\mathcal{D}_{\mathbf{x}}^B = \mathcal{D}_{\mathbf{x}}$. The relevant area is given as the curved section of a spherical cap C at distance d from the center for a sphere of radius q, using standard algebra, as

$$A(q, d) = 2\pi q(q - d), \tag{6.16}$$

which, again, is only well defined for $q \geq d$ and we, therefore, limit the integral to $[d, 1]$. Inserting (6.16) into (6.15) then yields

$$\int_d^1 \iint_0^{2\pi q(q-d)} C_3 \hat{W}(q) dA dq = \int_d^1 C_3 \hat{W}(q) 2\pi q(q - d) dq. \tag{6.17}$$

Analogously to 2D, we split the integral-based on the piecewise definition of the cubic spline kernel, and add an extension for $d < 0$ using rotational symmetry, yielding the definite integrals shown in the Appendix as (6.59). These can then be evaluated symbolically, yielding (6.60) in the Appendix, which put back into $\lambda_3(d)$ result in the boundary contribution term in 3D for a planar boundary as

$$\lambda_3(d) = \begin{cases} \frac{1}{60}\left[192d^6 - 288d^5 + 160d^3 - 84d + 30\right], & d \in [0, 0.5] \\ \frac{-8}{15}\left[2d^6 - 9d^5 + 15d^4 - 10d^3 + 3d - 1\right], & d \in (0.5, 1], \\ 1 - \lambda_3(-d), & d \in [-1, 0). \end{cases} \tag{6.18}$$

Interestingly, this term is significantly more simple than the equivalent term in 2D, see (6.57), but yields very similar numerical values.

6.5.3 Scaling factor

In order to extend the prior derivations to non-unit support radii we insert the definition of a generic kernel function from (6.3), into an integral over an arbitrary boundary domain $\mathcal{D}_{\mathbf{x}}^B$, which yields

$$\int_{\mathcal{D}_{\mathbf{x}}^B} W(r, h) d\mathbf{x}' = \int_{\mathcal{D}_{\mathbf{x}}^B} \frac{1}{h^n} C_n \hat{W}\left(\frac{r}{h}\right) d\mathbf{x}'. \tag{6.19}$$

As $\mathcal{D}_{\mathbf{x}}^B$ includes all radii from 0 to h, we perform a u-substitution with $q = r/h$, which yields

$$\int_{\mathcal{D}_{\mathbf{x}}^B} \frac{1}{h^n} C_n \hat{W}\left(\frac{r}{h}\right) d\mathbf{x}' = \int_{\mathcal{D}_{\mathbf{x}}^{B*}} C_d \hat{W}(q) d\mathbf{x}^*, \tag{6.20}$$

where the factor $\frac{1}{h^n}$ vanishes due to the substitution and $\mathcal{D}_{\mathbf{x}}^{B*}$ is the result of scaling $\mathcal{D}_{\mathbf{x}}^{B}$ to a radius of 1, which also means that the flat boundary is at a distance $d^* = d/h$ instead of d and $\mathbf{x}^* = \frac{\mathbf{x}'}{h}$.

In addition to the integral of the kernel function over the boundary domain, integral values of other functions may have to be calculated, e.g., the source term of a pressure solver like DFSPH [BK15]. Commonly, these functions depend on a radially symmetric function $f : \mathbb{R}_0^+ \to \mathbb{R}^m$, based on the relative distance r/h, divided by some power of the support radius, which yields an integral of the form

$$A_b \int_{\mathcal{D}_{\mathbf{x}}^{B}} \frac{1}{h^s} f\left(\frac{r}{h}\right) d\mathbf{x}'. \tag{6.21}$$

Here we can, again, apply a u-substitution to yield

$$A_b \int_{\mathcal{D}_{\mathbf{x}}^{B*}} \frac{1}{h^{s-n}} f(q) d\mathbf{x}^*. \tag{6.22}$$

6.5.4 Gradient terms

In addition to the boundary contribution term λ_n, related terms, i.e., the spatial derivative, ∇_i, with respect to a particle i, are required, for example for pressure forces. In order to calculate these gradients, we use the fact that the spatial derivative of the distance d for a flat boundary $\mathcal{D}_{\mathbf{x}}^{B}$ equals the normal of the flat boundary $\mathbf{n}_b$, which when normalized yields $\hat{n}_b = \frac{\nabla_i d}{||\nabla_i d||}$. Applying the chain rule gets

$$\nabla_i \left(A_b \lambda_n^b\right) = A_b \nabla_i \lambda_n^b\left(q\right) = A_b \frac{1}{h} \hat{n}_b \frac{\partial \lambda_n^b(q)}{\partial q}. \tag{6.23}$$

However, these direct gradient terms in SPH are not exact for constant functions and do not preserve symmetric interactions. By following the ideas of Price [Pri12] for particles, we can yield analogous terms for our integral-based boundary representation; see (6.66) and (6.67) in the Appendix. Applying $\frac{\partial}{\partial q}$ to $\lambda_3^b(q)$ yields

$$\frac{\partial \lambda_3(q)}{\partial q} = \begin{cases} \frac{3}{15}\left[96q^5 - 120q^4 + 40q^2 - 7\right], & q \in [0, 0.5] \\ \frac{-24}{15}\left[4q^5 - 15q^4 + 20q^3 - 10q^2 + 1\right], & q \in (0.5, 1], \\ -\frac{\partial}{\partial q}\lambda_3(-q), & q \in [-1, 0). \end{cases} \tag{6.24}$$

In order to realize two-way coupling, modern pressure solvers, see Bender and Koschier [BK15] Eqn. (8), evaluate a factor that, in terms of boundary integrals, is expressed as

$$\frac{\alpha_i}{\rho_0^2} = \left|\left|\int_{\Omega_{\mathbf{x}}} \nabla_i W(|\mathbf{x_i} - \mathbf{x}|, h) d\mathbf{x}\right|\right|^2 + \int_{\Omega_{\mathbf{x}}} ||\nabla_i W(|\mathbf{x_i} - \mathbf{x}|, h)||^2 d\mathbf{x}, \tag{6.25}$$

where the former term can be directly calculated using the gradient of the boundary contribution; see (6.23). To calculate the latter term, referred to as $\mathcal{S}$, we apply the spatial derivative ∇_i to (6.3), yielding

$$\nabla_i W(|\mathbf{x}_j - \mathbf{x}_i|, h) = \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|} \frac{1}{h^{n+1}} C_n \frac{\partial \hat{W}(q)}{\partial q}. \tag{6.26}$$

Consequently, we get

$$\mathcal{S}_n(d) = \int_{\mathcal{D}_{\mathbf{x}}^B} \frac{1}{h^{2n+2}} C_n^2 \left[\frac{\partial \hat{W}(q)}{\partial q}\right]^2 d\mathbf{x}', \tag{6.27}$$

Note that in contrast to λ_n in Eqn. (6.56) and (6.59), the integration of the function over the whole domain $\mathcal{D}_{\mathbf{x}}$ is not equal to 1. This results in the definite integrals (6.61), which can be symbolically integrated; see (6.62) in the Appendix. To correct for scaling, the result needs to be divided by h^{n+2} (Sec. 6.5.3), which results in a final term for $\mathcal{S}_3(d)$; see (6.63) in the Appendix. Similar steps could be taken to determine the Laplacian over a boundary, but these terms find no practical use in modern pressure solvers and as such are left out for brevity.

6.5.5 Penalty terms

A problem with the analytical solution proposed thus far is that for a particle that is fully within the boundary, i.e., $\mathcal{D}_{\mathbf{x}} = \mathcal{D}_{\mathbf{x}}^B$, the density gradient will be zero. This means that if a particle fully penetrates the rigid body it will never get repelled out of the rigid body. To avoid this problem Koschier and Bender [KB17] and Bender et al. [Ben+19] proposed different penalty functions, which are applied to the boundary contribution to penalize full penetration by artificially adding a monotonically increasing term onto the boundary terms. Similar to Koschier and Bender [KB17], we chose the following penalty function

$$\beta(d) = 1 - \frac{d}{h}, \frac{\partial \beta(d)}{\partial d} = -h^{-1}. \tag{6.28}$$

We directly multiply this penalty term onto $\lambda_n(d)$, which yields

$$\lambda_n^\beta(d) = \beta(d)\lambda_n(d), \tag{6.29}$$

which for gradients, by applying the product rule, yields

$$\nabla_i \lambda_n^\beta(d) = \frac{\partial \beta}{\partial d}\hat{n}_{\Omega_{\mathbf{x}}} \lambda_n(d) + \beta(d)\nabla_i \lambda_n(d). \tag{6.30}$$

6.6 Arbitrary boundary handling

To find a locally planar representation of a boundary, to which we can apply the analytic solution demonstrated in Sec. 6.5, we propose to utilize a signed distance field (SDF). Signed distance fields can be used to represent arbitrary geometries and yield the same signed distance value and gradient, regardless of particle resolution, making them well suited for adaptive methods. However, SDFs, in general, are not ideally suited for highly concave objects, as particles would only be influenced by one part of the geometry, i.e., they are limited to single contact points. This limitation, however, can be resolved by performing a convex body decomposition on the boundary geometry that allows multiple points of contact for a single particle with the same overall boundary geometry. Furthermore, *discretized* SDFs might not accurately represent the boundary geometry, e.g., due to aliasing artifacts. While Sec 6.9.5 discusses how this induced error can be measured, and

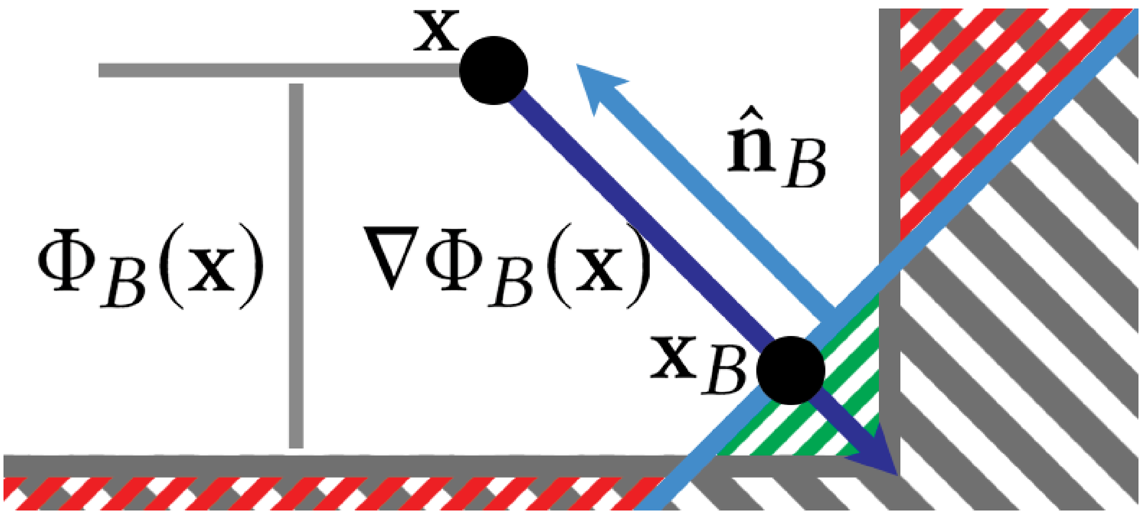


Figure 6.4: The construction of a local plane (blue) for a concave corner of a boundary (gray) based on the signed distance field Φ_B. This constructed plane ignores the contribution of some regions (red), but additionally includes other regions (green), causing an overall underestimate.

reduced. We assume a continuous SDF for this derivation, i.e., an SDF that is not limited in accuracy.

An SDF Φ_B is determined based on the surface of a boundary domain ∂B, for at an arbitrary spatial position $\mathbf{x}$, as

$$\Phi_B(\mathbf{x}) = s(\mathbf{x}) \inf_{\mathbf{x}^* \in \partial B} ||\mathbf{x} - \mathbf{x}^*||, s(\mathbf{x}) = \begin{cases} -1, & \mathbf{x} \in B \\ 1, & \text{else.} \end{cases} \tag{6.31}$$

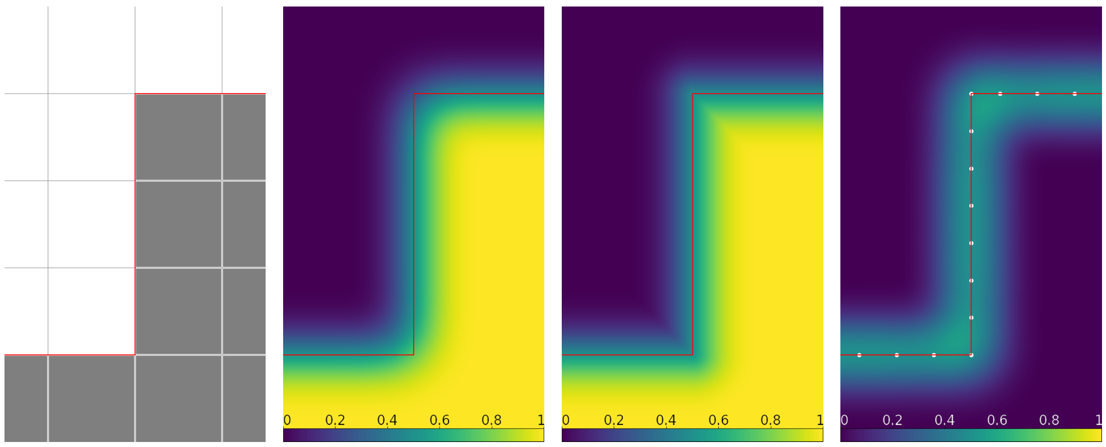
Using the normalized numerical gradient of the signed distance field, e.g., determined using a central difference scheme, we can find a unit normal $\hat{\mathbf{n}}_B = \nabla\Phi_B(\mathbf{x})/|\nabla\Phi_B(\mathbf{x})|$, which yields an approximate closest point on the boundary surface as

$$\mathbf{x}_B = \mathbf{x} - \Phi_B(\mathbf{x})\hat{\mathbf{n}}_B. \tag{6.32}$$

Note that the usage of a numerical gradient here results in a different result than using a continuous gradient. Fig. 6.4 shows how for a point $\mathbf{x}$ sitting on the angle dividing between two planes we find a numerical gradient pointing away from the corner, which also results in a contact point $\mathbf{x}_B$ that does not lie on the actual boundary geometry, but instead gives an approximative representation of the corner. Using $\hat{\mathbf{n}}_B$ and $\mathbf{x}_B$ we then construct a plane E_b, which represents the boundary b, in normal form

$$E_b : \hat{\mathbf{n}}_B \cdot (\mathbf{x} - \mathbf{x}_B) = 0, \tag{6.33}$$

where basing $\hat{\mathbf{n}}_B$ on the position $\mathbf{x}$ instead of $\mathbf{x}_B$ ensures that forces act away from the boundary. Figure 6.4 depicts this configuration for a convex corner, where the constructed plane causes an underestimate of the boundary contribution term. This underestimate, however, is unavoidable when trying to replace a complex boundary by a single plane. The effect of this underestimate is evaluated in the



(a) Boundary (b) Numerical (c) Distance-based (d) Particle-based

Figure 6.5: The boundary contribution term λ evaluated (a) using a numerical solution (b), our signed distance field-based solution (c) and a particle-based representation (d). The signed distance field-based solution represents the boundary geometry well, but shows underestimates in concave regions and overestimates in convex regions. The particle-based approach cannot distinguish between inside and outside of the boundary and shows similar overestimates in convex regions and underestimates in concave regions. Note that the particle-based approach only represents the surface of the boundary, i.e., there is no way to distinguish interior and exterior of the boundary. λ color coded from 0 (purple) to 1 (yellow).

results section. For a concave configuration the approximative boundary plane causes an overestimate of the boundary contribution term. Fig. 6.5 shows the boundary density contribution for a simple boundary (Fig. 6.5a), using the 30th order quadrature rule of Xiao and Gambutas [XG10] for a triangulated boundary domain (Fig. 6.5b), our signed distance field-based approach (Fig. 6.5c), and a particle-based approach (Fig. 6.5d). The particle-based approach also shows over and underestimates for the contribution and provides distinction between the outside and the inside of the boundary, causing severe problems for fluid particles penetrating the boundary slightly.

For moving boundary objects, a recalculation of the signed distance field in every simulation step would be far too expensive. Therefore, we use a rigid body transformation matrix $\mathbf{M}_B$ from local model coordinates, in which the origin is the object's center of mass, to global simulation coordinates. Using $\mathbf{M}_B$ any point $\mathbf{x}$, in global fluid space, can be transformed, into the local model space, via

$$\mathbf{x}^* = \mathbf{M}_B^{-1}\mathbf{x}. \tag{6.34}$$

Afterwards, the closest point on the boundary $\mathbf{x}_B^*$ is constructed in model space and transformed back to global simulation coordinates using $\mathbf{M}_B$. The corresponding normal of the plane $\hat{\mathbf{n}}_B^*$ is calculated using the transposed inverse transformation $\mathbf{M}^{-T}$, yielding

$$E_b : \left(\mathbf{M}_B^{-T}\hat{\mathbf{n}}_B^*\right) \cdot \left(\mathbf{x} - \mathbf{M}_B\mathbf{x}_B^*\right) = 0, \tag{6.35}$$

as the representative plane with the contact point $\mathbf{M}_B\mathbf{x}_B^*$. This process is also depicted in Algorithm 6.1. We also utilize this process in case of multiple boundary objects, i.e., each boundary object is represented by an indepedent SDF in

ALGORITHM 6.1: The process to find a locally representative plane for a complex boundary object B using its signed distance field in model coordinates.

```
1 $\mathbf{x}^* \leftarrow \mathbf{M}_{\mathcal{B}}^{-1}\mathbf{x}$
2 $d \leftarrow \Phi_B^*(\mathbf{x}^*)$
3 If $d > h$
4     Return
5 $\hat{\mathbf{n}}_B^* \leftarrow= -\nabla\Phi^*(\mathbf{x}^*)/|\nabla\Phi^*(\mathbf{x}^*)|$
6 $\mathbf{x}_B^* \leftarrow \mathbf{x}^* - \Phi^*(\mathbf{x}^*)\hat{\mathbf{n}}_B^*$
7 Return the representative plane (6.35)
```

the respective objects model coordinates, and interacting with multiple boundary objects only requires a transformation into each boundary objects coordinate system. Accordingly, our approach can readily handle multiple boundary objects without requiring complex SDF operations, i.e., finding a global SDF describing all objects in a scene.

In order to implement two-way coupling into our simulation, we require several properties of the rigid body b based on the contact point $\mathbf{x}_{ib}$ associated with the position of a fluid particle i. Therefore, we first determine the center of mass of the rigid body in simulation space as $\mathbf{x}_b^c = \mathbf{M}_b\mathbf{0}$, which yields the velocity of the contact point as

$$\mathbf{v_{ib}} = \mathbf{v}_b + \omega_b \times \left(\mathbf{x}_i - \mathbf{x}_b^c\right), \tag{6.36}$$

where $\mathbf{v}_b$ and ω_b are the linear and angular velocity of the overall boundary object. The acceleration $\mathbf{a_{ib}}$ of the contact point can similarly be calculated. Determining the influence of a boundary object on a particle can be done straightforwardly by including the boundary contribution term for a particle (see Eq. 6.10), however determining the influence of particles on the boundary cannot be done directly. Instead, we propose to add up the inverse force from a boundary, $\mathbf{F}_{i\to b} = -\mathbf{F}_{b\to i}$, due to all particles, which yields the influence of the fluid on the boundary. In addition to the force $\mathbf{F}_{i\to b}$ we also require the torque as

$$\tau_{i\to B} = m_i\mathbf{a}_{i\to B} \times \left[\mathbf{x} - \mathbf{x}_c\right]. \tag{6.37}$$

From this, we determine the linear and angular acceleration of the overall boundary object, based on the inertia matrix $\mathbf{I}_b^*$ in model coordinates as

$$a_{i\to B} = \mathbf{F}_{i\to B}/m_B, \alpha_{i\to B} = \mathbf{M}_B\left(\mathbf{I}^*\right)_B^{-1}\mathbf{M}_B^{-1}\tau_{i\to B}. \tag{6.38}$$

Finally, for implementing our method, the maximum permissible timestep needs to be considered. We utilize the CFL condition [Ihm+14], excluding shock terms, as

$$\Delta t^{\text{max}} = 0.4\text{max}_i\frac{||\mathbf{v}_i||}{h_i}. \tag{6.39}$$

For our semi-analytical approach this is not directly possible as there are no boundary particles that we can use (6.39) for. Instead, we utilize the point furthest from the axis of rotation of a boundary object b, $\mathbf{x}_{\text{b,max}}$, in order to determine the maximum local velocity of the boundary object as

$$\mathbf{v}_{\text{b,max}} = \mathbf{v}_b + \omega_b \times \left[\mathbf{x}_{\text{b,max}} - \mathbf{x}_b^c\right], \tag{6.40}$$

based on which we can determine the maximum permissible timestep analogous to Eq. (6.39). However, instead of the support radius for the particle, we use the smallest support radius of any particle in the simulation to ensure stability in all cases.

6.7 Integration into DFSPH

In this section we discuss the integration of our boundary integral method with a divergence-free SPH method based on Bender and Koschier [BK15] and Band et al. [Ban+18b]; see Algorithm 6.2. The first step of the simulation comprises the calculation of the density value for each fluid particle i including boundary contributions, utilizing (6.10), as

$$\rho_i = \sum_j m_j W_{ij} + \sum_b \rho_b \lambda^b_{i,n}. \tag{6.41}$$

For boundary objects we assume that ρ_b is equal to the object's respective rest density, as we only allow rigid body transformations, which only change the position of an object but not its shape. Note that V_i here refers to the apparent volume of a fluid particle

$$V_i = \frac{m_i}{\rho_i}, \tag{6.42}$$

instead of the actual volume $V_i^0 = \frac{m_i}{\rho_0}$; see Band et al. [Ban+18a]. We also require a predicted velocity, $\mathbf{v}_i^* = \mathbf{v}_i + \frac{\Delta t}{m_i}\mathbf{F}_i^{\text{adv}}$, after taking advection forces into account. In order to calculate this predicted velocity for the boundary objects, we utilize standard rigid body physics to find the specific local boundary velocity $\mathbf{v}^*_{ib}$ for a particle, which depends on the particle's contact point $\mathbf{x}_{ib}$ at the boundary. Next we need to determine the diagonal element α_i [Ban+18b] as

$$\begin{aligned} \alpha_i = &- V_i \Delta t^2 \frac{1}{m_i} || \sum_j V_j \nabla_i W_{ij} + \sum_b \nabla \lambda^b_{i,n} ||^2 \\ &- V_i \Delta t^2 \sum_j \frac{V_j^2}{m_j} ||\nabla_i W_{ij}||^2 - V_i \Delta t^2 \sum_b \frac{1}{\rho_b} \mathcal{S}^b_{i,n}, \end{aligned} \tag{6.43}$$

where $\mathcal{S}^b_n$ is the given in Eq. (6.63) for $n = 3$. For all non two-way coupled and static boundary objects $\mathcal{S}^b_n$ is set to 0, as no pressure forces are applied to these boundaries; see also [Ban+18a]. Similar to Band et al. [Ban+18b], we determine the source terms for the divergence-free and incompressible solver as

$$s_i^{\text{inc}} = 1 - \frac{m_i}{V_i} - \Delta t \nabla_i \cdot \mathbf{v}^*, \quad s_i^{\text{div}} = -\Delta t \nabla_i \cdot \mathbf{v}, \tag{6.44}$$

respectively. The velocity divergence can be calculated as

$$\nabla_i \cdot \mathbf{v} = \sum_j V_j \Delta \mathbf{v}_{ij} \cdot \nabla_i W_{ij} + \sum_b \Delta \mathbf{v}_{ib} \cdot \nabla \lambda^b_{i,n}, \tag{6.45}$$

where $\Delta \mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. Note that we only enforce the divergence-freedom for fluid-fluid interactions to avoid excessive adhesion and repulsion as a particle separating from a boundary would cause negative divergence, i.e., its density becomes lower, and a particle approaching a boundary would cause positive divergence. This is because we cannot evaluate the opposite effect on the boundary, i.e., when a particle has positive divergence the boundary object would experience negative divergence, during the divergence-free solving process. Other boundary-integral methods, e.g., [KB17], would be affected by the same problem. To initialize the pressure, we follow the proposed initialization of Gissler et al. [Gis+19] and set $p_i = 0$ for all fluid particles. The next step is to determine the acceleration of a fluid particle due to pressure forces, due to fluid-fluid interactions, as [BK15]

$$a^p_{\text{fluid}\to i} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla_i W_{ij}. \tag{6.46}$$

Using the same formulation to calculate the pressure acceleration imposed by rigid objects requires a pressure value p_{ib} for the boundary. This pressure value is calculated by determining the contact point $\mathbf{x}_{ib}$ on the boundary and performing the moving-least-squares-based pressure extrapolation method proposed by Band et al. [Ban+18b]. As we assume quantities to be locally constant, we can directly use the pressure value p_{ib} for the whole interaction. Thus, the total acceleration imposed on a fluid particle i by all rigid objects can be determined as

$$\mathbf{a}^p_{\text{rigid}\to i} = -\sum_b \rho_b \left(\frac{p_i}{\rho_i^2} + \frac{p_{ib}}{\rho_b^2} \right) \nabla_i \lambda^b_{i,n}. \tag{6.47}$$

We additionally utilize the acceleration imposed on a fluid particle by a single rigid object b, $\mathbf{a}^p_{b\to i}$, to calculate the reflected force on b, $\mathbf{F}^p_{i\to b} = -m_i \mathbf{a}^p_{b\to i}$, to realize two-way coupling effects. Accumulating the coupled force imposed by all particles yields the acceleration of the rigid object as a whole, which yields the local boundary acceleration $\mathbf{a}^p_{ib}$, at the contact point with respect to i. Moreover, we calculate the divergence of the estimated acceleration as

$$\nabla_i \cdot a^p_i = \Delta t^2 \sum_j V_j \mathbf{a}^p_{ij} \cdot \nabla_i W_{ij} + \Delta t^2 \sum_b \mathbf{a}^p_{ib} \cdot \nabla \lambda^b_{i,n}, \tag{6.48}$$

and an updated pressure value for each fluid particle as

$$p^*_i = p_i + \frac{\omega}{\alpha_i} \left(s_i - \nabla_i \cdot a^p_i \right), \tag{6.49}$$

where ω is the relaxation coefficient for a relaxed Jacobi solver, chosen as 0.5. We clamp the pressure to positive values for the incompressible solver, to avoid excessive surface-tension like effects. This process is repeated until the mass weighted average of the residual, $r_i = \nabla_i \cdot a^p_i - s_i$, converges below a threshold η, i.e.,

$$\sum_i r_i m_i \leq \eta \sum_i m_i, \tag{6.50}$$

where η is usually chosen as 0.0001 for the incompressible solver and 0.001 for the divergence-free solver, similar to Bender and Koschier [BK15]. After the solver converges, we calculate the pressure acceleration and apply it to both fluid particles and boundary objects. We finally integrate the system accordingly.

ALGORITHM 6.2: Our overall simulation algorithm using a divergence-free and incompressible simulation approach with two-way coupling for complex boundary objects.

```
1  $\rho_i \leftarrow \sum_j m_j W_{ij} + \sum_b \rho_b \lambda_n^b;\quad V_i \leftarrow \frac{m_i}{\rho_i}$ (6.41, 6.42)
2  **Calculate** Diagonal element $\alpha_i$ (6.43)
3  $p_i \leftarrow 0;\quad s_i^{\text{div}} \leftarrow -\Delta t \nabla_i \cdot \mathbf{v}$ (6.44)
4  **while** $\sum_i m_i \nabla_i \cdot \mathbf{a}_i^p - s_i^{\text{div}} > \eta^{\text{div}} \sum_i m_i$ (6.50)
5      **Calculate** Boundary pressure $p_{ib}$ using MLS
6      **Calculate** Acceleration due to pressure (6.47,6.46)
7      $p_i^* \leftarrow p_i + \frac{\omega}{\alpha_i}\left(s_i - \nabla_i \cdot a_i^p\right)$, (6.49)
8  **Calculate** Final acceleration due to pressure $\mathbf{a}_i^{\text{div}}$ (6.47,6.46)
9  **Calculate** Non-pressure acceleration $\mathbf{a}_i^{\text{adv}}$
10 $\mathbf{v}_i^* \leftarrow \mathbf{v}_i + \Delta t\left[\mathbf{a}_i^{\text{adv}} + \mathbf{a}_i^{\text{div}}\right]$
11 $p_i \leftarrow 0;\quad s_i^{\text{inc}} \leftarrow 1 - \frac{m_i}{V_i} - \Delta t \nabla_i \cdot \mathbf{v}^*$ (6.44)
12 **while** $\sum_i m_i \nabla_i \cdot \mathbf{a}_i^p - s_i^{\text{inc}} > \eta^{\text{inc}} \sum_i m_i$ (6.50)
13     **Calculate** Boundary pressure $p_{ib}$ using MLS
14     **Calculate** Acceleration due to pressure (6.47,6.46)
15     **Calculate** Acceleration of boundaries; see Sec. 6.6
16     $p_i^* \leftarrow p_i + \frac{\omega}{\alpha_i}\left(s_i - \nabla_i \cdot a_i^p\right)$, (6.49)
17 **Calculate** Final acceleration due to pressure $\mathbf{a}_i^{\text{p}}$ (6.47,6.46)
18 **Calculate** Acceleration due to friction $\mathbf{a}_i^c$ (6.55)
19 $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\left[\mathbf{a}_i^p + \mathbf{a}_i^{\text{div}} + \mathbf{a}_i^c + \mathbf{a}_i^{\text{adv}}\right];\quad \mathbf{x}_i \leftarrow \mathbf{x}_i \Delta t \mathbf{v}_i$
20 **Update** Rigid body system; see Sec. 6.6
```

6.8 Friction

To model fluid-rigid friction effects Koschier and Bender [KB17] proposed a Coulomb model based force, which we also adopt for our approach as it is based on the same underlying boundary-integral model, however our method can also be applied to other friction models, e.g., [RDS10]. The friction model of Koschier and Bender [KB17], however, does not consider boundary pressure values. The Coulomb model determines a friction force for the interaction of two bodies i and b as

$$\mathbf{F}_{b\to i}^c \leq \mu ||\mathbf{F}_{i\to b}^n|| \hat{\mathbf{t}}_{ib}, \tag{6.51}$$

where μ is the friction coefficient, $||F_{i\to b}^n||$ the force imposed on b by i in normal direction and $\hat{\mathbf{t}}_{ib}$ the direction of the relative tangential velocity of i and b. The tangential velocity can be determined as

$$\mathbf{t}_{ib} = \Delta \mathbf{v}_{ib} - \left(\Delta \mathbf{v}_{ib} \cdot \hat{\mathbf{n}}_b\right) \cdot \hat{\mathbf{n}}_b, \tag{6.52}$$

with the relative velocity $\Delta \mathbf{v}_{ib} = \mathbf{v}_i - \mathbf{v}_b$. For the normal force we simply consider the pressure force, including the estimated boundary pressure of the rigid body, as

$$||\mathbf{F}_{b\to i}^p|| = -m_i \rho_b \left(\frac{p_i}{\rho_i^2} + \frac{p_{ib}}{\rho_b^2}\right) ||\nabla \lambda_{i,n}^b||. \tag{6.53}$$

The acceleration caused by all rigid bodies b on i then becomes

$$\mathbf{a}'_{b\to i} = \frac{1}{m_i}\sum_b \mathbf{F}^c_{b\to i} \tag{6.54}$$

The velocity change due to this imposed acceleration, however, can become greater than the relative velocity, which would cause unphysical behavior. To exclude such cases we limit the imposed acceleration to at most $-\frac{1}{\Delta t}\mathbf{t}_{ib}$ if $\Delta t\mathbf{a}'_{b\to i}\cdot\hat{\mathbf{t}}_{ib} \geq \Delta\mathbf{v}_{ib}\cdot\hat{\mathbf{t}}_{ib}$, which yields the final friction acceleration term as

$$\mathbf{a}^c_{b\to i} = \frac{1}{m_i}\sum_b \begin{cases}\mathbf{a}'_{b\to i}, & \Delta t\mathbf{a}'_{b\to i}\cdot\hat{\mathbf{t}}_{ib} < \Delta\mathbf{v}_{ib}\cdot\hat{\mathbf{t}}_{ib}\\ -\frac{1}{\Delta t}\mathbf{t}_{ib}, & \text{else.}\end{cases} \tag{6.55}$$

6.9 Evaluation

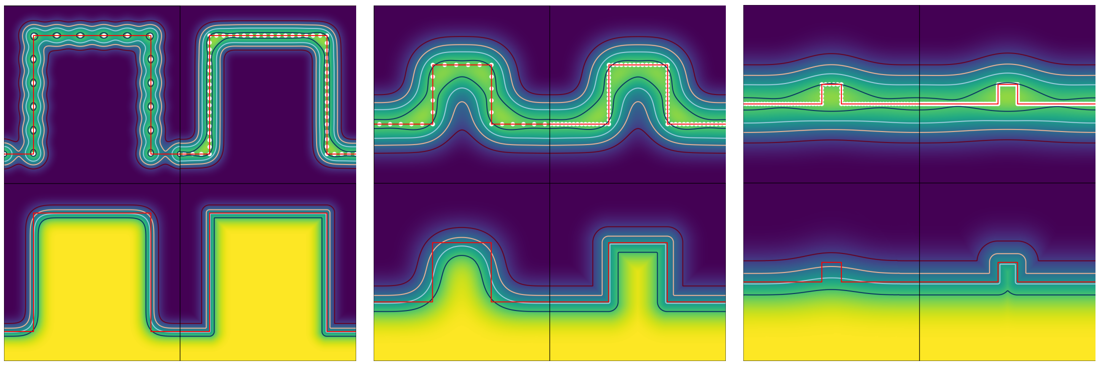
In this section, we evaluate our method by comparing it against other boundary representations for small-sized boundary features in 2D (Sec. 6.9.2- 6.9.4), one-way and two-way coupled simulations in 3D (Sec. 6.9.6), for different boundary samplings (Sec. 6.9.7), as well as regarding adaptivity, friction and stability (Sec. 6.9.8-6.9.10).

6.9.1 Simulation setup

We implemented our method in the open source GPU-based SPH frameworkopen-Maelstrom [Win19], using the adaptive method of Winchenbach et al. [WHK17], XSPH based artificial viscosity [Mon02], fluid-air phase interactions based on Gissler et al. [Gis+17], surface tension based on Akinci et al. [AAT13], adaptive time-stepping based on the CFL condition [Ihm+13], the underlying data structure of Winchenbach and Kolb [WK19] and the cubic spline kernel [Mon05]. We used an incompressibility threshold of 0.01% and a divergence threshold of 0.1%, similar to Bender and Koschier [BK15]. Surface distance calculations, for adaptivity, are based on Horvath and Solenthaler [HS13]. For comparisons with particle based methods we used ILSPH [Gis+19] with the optimized particle sampling of Bell et al. [BYM05] in 3D and the approach of Akinci et al. [Aki+13b] with a regular sampling in 2D. The numerical boundary-integral method in 2D is realized by triangulating the boundary and using the quadrature rule of Xiao and Gambutas [XG10] (at 30th order) to evaluate (6.2) over each triangle. The resulting images, utilizing this framework, were rendered using a Monte-Carlo based path-tracing algorithm. In our results frame-time refers to the accumulated runtime to simulate 16.67ms of simulation-time. For the comparison with the volume maps approach [Ben+19], we utilized the open source SPH framework SPlisHSPlasH [Ben16], using the same methods for fluid effects, e.g., surface tension, running on a CPU instead of on a GPU, and resulting images rendered using Autodesk Maya. All results were generated using a single Nvidia GeForce RTX 2080Ti GPU with 11 GiB of VRAM, an AMD 3970X, and 64 GiB of RAM.

Table 6.1: Quantitative comparison. All performance numbers are average values with respect to 30 seconds simulation time and timings refer $1/60$s of simulation time. The radius values refers to the largest fluid particle radius in the simulation, and the ratio refers to the volume ratio of the largest to smallest particle volume. Incomp. and Div.-Free refer to the incompressible and divergence-free solver, respectively. In all results the divergence-free solver required 2 iterations.

Scene	**Figure**	n_{fluid}	n_{rigid}	**radius** $/m$	**ratio**	**timestep** $/ms$	**Incomp.** Iterations	**Frame-time** $/ms$	**Div.-Free** $/ms$	**Incomp.** $/ms$	**Adapt** $/ms$
Basin	6.14b	23K		0.5		8.0	2.86	60	5	13	
Basin	6.14c	23K	81K	0.5		8.0	6.48	74	4	8	
Basin	6.14d	3.17M		0.1		2.0	4.6	2661	262	1218	
Basin	6.14e	3.17M	2.05M	0.1		1.9	9.5	5231	392	2263	
Dragon	6.10 left	133K	183K	0.5		8.0	12.58	146	14	53	
Dragon	6.10 middle	133K		0.5		7.5	15.87	384	32	225	
Dragon	6.10 right	133K		0.5		8.0	10.71	94	7	53	
Dragon	6.11	1.98M		0.5	1000	3.9	7.2	2301	105	310	422
Propeller		539K		0.3		3.4	41.5	1642	40	1081	
Propeller	6.1	1.98M		0.3	100	0.7	3.1	9501	347	871	1321
Incline	6.17 left	56K		0.5		8.0	8.0	112	7	35	
Incline	6.17 right	56K		0.5		8.0	8.1	111	6	35	
Boxes	6.12	196K		0.5		6.0	9.3	267	12	113	
Boxes	6.13	1.65M		0.5	500	3.3	7.5	3561	116	376	450
Boxes		196K	124K	0.5		6.0	24.3	532	11	367	



(a) $1:4$ ratio (b) $1:1$ ratio (c) $4:1$ ratio

Figure 6.6: A numerical evaluation of how different boundary handling schemes can resolve a fixed size object (red, 1×1 unit) with different ratios of support radius to object size. For each variant the top row shows a particle-based method with a coarse (left) and fine (right) sampling and the bottom row shows a numerical boundary integral method (left) and our SDF-based method (right). The boundary contribution term λ is color coded from 0 (purple) to 1 (yellow). The isocontours show where a lone particle (black) and particles with quarter (blue), half (tan) and three quarter (purple) full neighborhoods would rest.

In 2D we compute SDFs directly, without sampling, based on a polygon representing the boundary domain. In 3D SDFs can also be evaluated on the fly, based on a triangular mesh, but this is usually computationally prohibitive. Instead, we precompute a discrete hierarchical SDF using OpenVDB [Mus+13], based on the boundary mesh, and then transfer the SDF onto a GPU using GVDB [Hoe16]. The process of constructing this SDF, and finding an appropriate grid resolution, is discussed in Sec. 6.9.5. The spatial gradients of the SDFs are determined using a second order central difference scheme. If this results in a zero length gradient, we instead utilize a first order forward difference scheme.

The matrix inverse for the moving-least-squares-based pressure estimation process [Ban+18a] is evaluated using a singular value decomposition (SVD). In 2D we use a closed form solution for the SVD of a 2×2 matrix and the optimized SVD process for 3×3 matrices of McAdams et al. [McA+11] in 3D. The implementation of the 3D process is available as opensource code [Win18].

6.9.2 Boundary representation

We compare our approach to a numerical boundary integral and a particle-based approach with a coarse and a fine boundary sampling, regarding their ability to represent a fixed size boundary feature with 1:4, 1:1 and 4:1 ratio of support radius to boundary feature size. We, therefore, compute isocontours where particles with no neighbors and quarter, half and three quarter full neighborhoods are located. For the fluid to interact correctly with a boundary feature, the isocontours of the fluid particles have to follow the geometry of the feature and must not intersect it. In this comparison the numerical boundary-integral approach yields identical results to a wall-renormalization approach.

For a 1:4 ratio (see Fig. 6.6 left), our approach and the boundary integral approach yield identical results in planar-regions h away from any corner. Close to corners, however, our approach generates significantly sharper isocontours, whereas the boundary-integral yields smoothed out corners. The particle-based approach, at a coarse sampling, cannot represent this ratio as the boundary would be penetrable for isolated particles, whereas a fine sampling of the boundary is impenetrable. Additionally, both samplings generate uneven isocontours and particles sit at a significant offset from the actual boundary geometry.

For a 1:1 ratio (see Fig. 6.6 middle), our approach preserves the vertical sections of the boundary geometry, for all isocontours, whereas the boundary-integral approach significantly smoothens out the boundary geometry. Our approach preserves the geometry as, by construction, the isocontour for a particle of density 0.5 is coincidental with the boundary geometry, which ensures that the transition from outside to inside the boundary is always centered around the actual geometry. The particle-based approach is impenetrable and yields similar results for both sampling densities, but also smoothens out the boundary geometry and still exhibits uneven isocontours.

For a 4:1 ratio (see Fig. 6.6 right), both the boundary-integral approach and the particle-based approach smooth out the feature extremely, while our approach still resolves the boundary geometry properly for particles with at least a quarter full neighborhood.

6.9.3 Penalty functions

There still remains a problem of an imprecise recovery of the location of isolated particles for high particle-to-feature size ratios (see the black isocontour for the 4:1 ratio, Fig 6.6 top rightmost). As our approach centers the boundary transition on the surface, the distance at which particles, with certain densities, rest is constant with respect to the surface of the boundary. Accordingly, any boundary feature smaller than twice this separation distance for isolated particles can be penetrated. This issue can be addressed by the idea of penalty functions, which modify the contribution of the boundary based on the distance to the boundary; see Sec. 6.5.5. Geometrically speaking, penalty functions shift and shrink the isocontours relative to the boundary geometry.

We now compare our approach with no penalty function, with the linear penalty function of Koschier and Bender [KB17], the cubic penalty function of Bender et al. [Ben+19] and with a modified softmax function $\beta(d) = \frac{\ln\left(1+e^{-2.5d}\right)-\ln(1+e^{-2.5})}{\ln(2)}$; see Fig. 6.7. The cubic penalty function is equal to 1 on the boundary interior and, as such, fails to resolve the penetration issue and instead only compresses isocontours on the outside. The linear penalty function, and the softmax function, achieve an isocontour for isolated particles that follows the feature outline. As previously discussed by Bender et al. [Ben+19], the linear penalty function is not differentiable at distance h from the boundary, which might cause undesirable behavior in a simulation. In general, the softmax penalty function, which is continuously differentiable for any distance, allows for the interaction of particles with very small obstacles, i.e., with features that are 20 times smaller than the support radius. In practice, we utilize the linear penalty function, as it allows for the interaction with small boundary features, but does not incur the same additional computational cost.

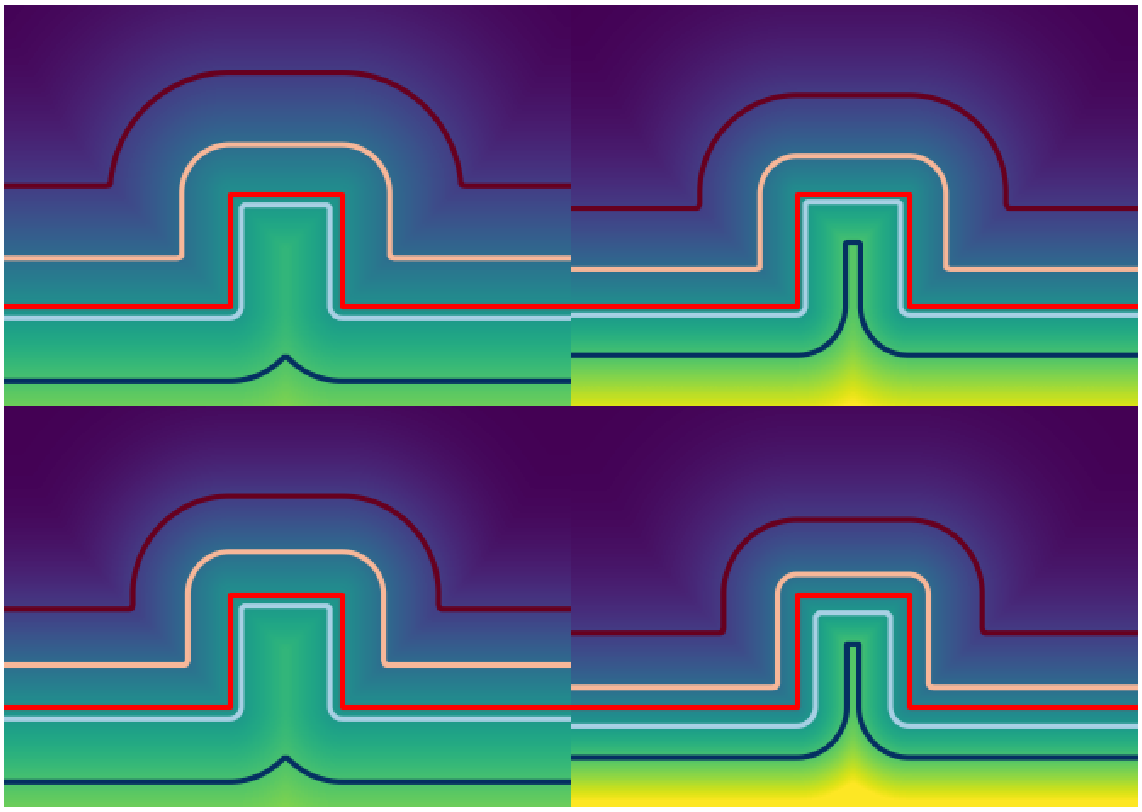


Figure 6.7: Comparison of different penalty functions for the 4:1 ratio of Fig. 6.6. Here, the top left uses no penalty function, top right uses the penalty function of Koschier and Bender [KB17], bottom left uses the penalty function of Bender et al. [Ben+19], and bottom right uses a modified softmax function. λ color coded from 0 (purple) to 1 (yellow).The penalty functions significantly impact the representaiton of a small feature, especially for lone particles (black isocontour), i.e., the obstacle might only be repsented as a bump.

6.9.4 Simulating small boundary features in 2D

To evaluate the impact of the difference in boundary representation further, we compare the difference of our approach and the numerical boundary-integral approach in a 2D dambreak scenario with an obstacle of fixed size (1×1 unit) and particles with relative support radii of 2, 1, 1/2 and 1/4; see Fig. 6.8. Considering our approach (top row), we can observe a similar deflection angle and overall behavior regardless of particle resolution, barring a variation of about 2 particle radii, with no penetration of the boundary object at any resolution. However, our method, at any resolution, has a single particle sitting on the bottom left vertex of the obstacle, which will be further investigated later. Considering the boundary-integral approach (Fig. 6.8 bottom row), we do not observe a consistent deflection angle and significant penetration of the boundary feature, especially at the 2:1 ratio. Moreover, the deflected flow becomes more collimated and the deflection angle gets steeper for finer particle resolutions due to the boundary becoming less smoothed out; see also Sec. 6.9.2. Only for even higher particle resolutions, i.e., 1/64th of the feature size, the deflection angle converges and coincides with the one in our approach. However, these high resolutions require inappropriatly high computational resources. Our approach preserves the overall behavior and yields

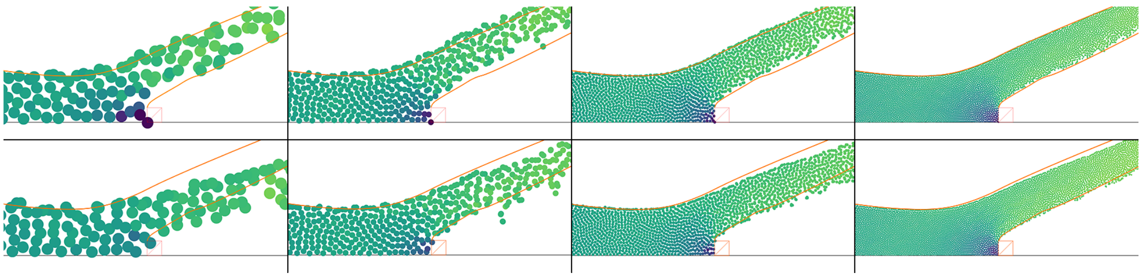


Figure 6.8: Comparison of our semi-analytical approach (top row) to a numerical boundary-integral approach (bottom row) for an obstacle of fixed size (1×1 units) and particles with support radius 2, 1, 1/2 and 1/4 (from left to right). The orange outline superimposes the outline of the plume of the top right variant to all other variants. Velocity color coded from purple ($0m/s$) to yellow ($25m/s$).

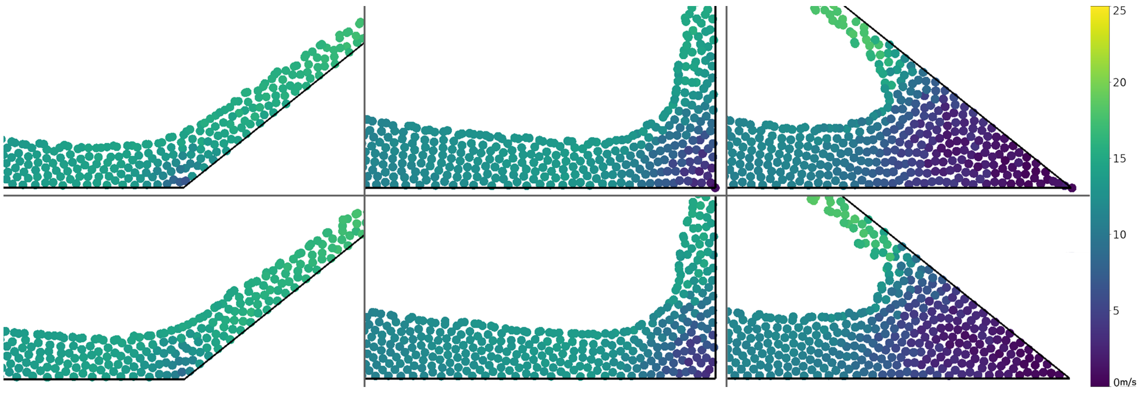


Figure 6.9: Comparison of our semi-analytical approach (top row) to a numerical approach (bottom row) for an obtuse (left), an orthogonal (middle) and an acute angle (right) in 2D. Our semi-analytical solution provides very similar overall flow behavior, except for a single particle resting on the actual corner. Velocity color coded from purple ($0m/s$) to yellow ($25m/s$).

very consistent deflection angles across a wide range of particle resolutions and, thus, resolves significantly smaller boundary geometry than prior methods.

To investigate this effect we evaluate the influence of the boundary sharpness by varying the angle of the boundary in a corner scene from acute to obtuse; see Fig. 6.9. In all three cases our approach yields nearly identical fluid behavior on the fluid surface. Near to the corner, however, as the corner angle becomes more acute, a single particle becomes trapped close to the corner. Whilst this effect does not introduce artifacts into the overall flow behavior, it can be visually apparent. We, therefore, avoid this effect on orthogonal simulation domain boundaries by rounding off the corners.

6.9.5 Discretized signed distance fields

While a continuous SDF can represent arbitrary geometry exactly, discretizing the SDF commonly yields a loss of fine details and may introduce sampling artifacts, e.g., aliasing effects. We chose to represent SDFs as a hierarchical sparse voxel structure using OpenVDB [Mus+13] and GVDB [Hoe16], where the resolution of the SDF is controlled by the voxel size. To determine the ideal voxel size to represent a given boundary mesh, we performed the process shown in Algorithm 6.3 to deter-

ALGORITHM 6.3: Given an input boundary mesh $\mathcal{M}$, this procedure finds the appropriate discrete SDF $\mathcal{B}$ that represents the input boundary geometry to within an error threshold η.

```
1 $i \leftarrow 0$ Do
2     $r \leftarrow r_{\text{base}} \left(\frac{1}{2}\right)^i ; i \leftarrow i + 1$
3     Generate candidate SDF $\mathcal{B}'$ with resolution $r$
4     Reconstruct the isosurface of $\mathcal{B}'$, yielding a mesh $\mathcal{M}'$
5     Evaluate the mesh difference $D = \text{dist}(\mathcal{M}, \mathcal{M}')$
6     $d \leftarrow \sup_{d' \in D} d'$
7 while d $> \eta$
8 Return SDF $\mathcal{B}'$
```

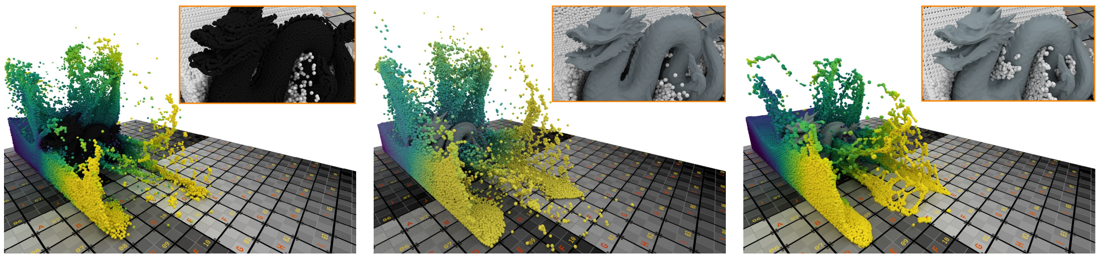

(a) ILSPH [Gis+19] (b) Volume Maps [Ben+19] (c) Our approach

Figure 6.10: Simulation of a fluid volume hitting a complex boundary object (Stanford dragon) with different boundary handling approaches. Velocity color coded from purple ($0m/s$) to yellow ($30m/s$) for the main view. The closeups show an earlier timepoint with fluid particles as white and boundary particles as black.

mine the best voxel size iteratively. In this process a candidate SDF is generated for a given voxel size and the mesh distance of a mesh reconstructed from this SDF and the initial boundary mesh is computed. Based on this mesh distance we then either further reduce the voxel size or stop the process if a certain error threshold is met.

As the error threshold we evaluated the maximum mesh distance and checked if this distance is less than the smallest particle radius expected within the simulations. This threshold results in a ratio of support radius, determined as $h_i = \sqrt[3]{50} r_i \approx 3.68 r_i$ [WHK16], to boundary accuracy of at least $3.68 : 1$. For the propeller scenario (Fig. 6.1) the maximum mesh distance was $4.45 \cdot 10^{-2}$, with a mean distance of $2.7 \cdot 10^{-4}$, for a finest particle radius of $6.46 \cdot 10^{-2}$, with an according smallest support radius of $2.38 \cdot 10^{-1}$, resulting in a maximum ratio of, approximately, $5.35 : 1$. Note that a lower voxel resolution could be utilized if the maximum error is constrained to spatial regions that are not important, e.g., if no fluid particles reach these regions, as the mean error is usually significant lower than the maximum. Furthermore, adaptive SDF representations, e.g., [KDB16], could also be utilized to adjust the error locally based on the complexity of the geometry, but in our scenarios this was not necessary.

6.9.6 Comparison of boundary handling in 3D

We first compare our method with Interlinked SPH [Gis+19] in the one-way cou-

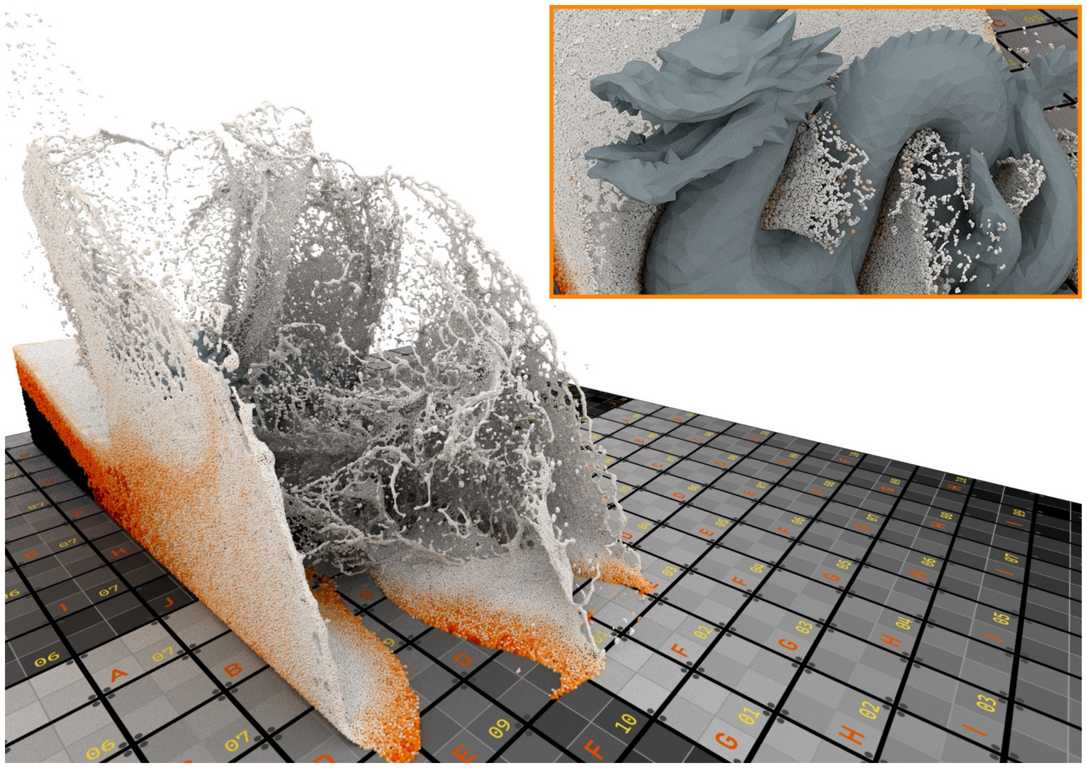


Figure 6.11: Adaptive variant of the *Dragon* scene using our boundary handling approach and particle volume color coded from black (high) to white (low).

pled *Dragon* scene, where a fluid volume impacts a static Stanford dragon model; see Fig. 6.10. Here, we observe an overall speedup of $1.5\times$ times for our approach when compared to ILSPH; see Tab. 6.1. In more detail, the lower overall particle count (no boundary particles) reduces the cost of, for example, the neighborlist creation and the moving-least-squares pressure extrapolation reduces the overall iteration count, however each individual iteration is computationally slightly more expensive. Moreover, the simulation using the volume maps approach [Ben+19] also shows a higher overall iteration count, due to the lack of a pressure extrapolation approach. However, note that the overall timings are not comparable, as the volume maps approach is simulated on a CPU instead of on a GPU. Furthermore, there are significant visual differences between the three simulation approaches. The main difference is due to the fluid not flowing through some openings of the dragon; see the close-ups in Fig. 6.10. For the volume maps approach this issue arises due to a strong dependence on the sampling resolution of the boundary geometry, which in this case was chosen equal to the sampling resolution used in [Ben+19] for the dragon model. Increasing the sampling resolution, i.e., by a factor of 16 in each direction, can reduce some of these sampling deficiencies but requires approximately 2 hours for precomputing the volume map and over 50GB of memory to store it, making it difficult to use, especially on GPUs. Furthermore, the volume, and density, map construction processes also involve first evaluating an SDF that represents the boundary geometry and, accordingly, are also affected from issues inherent to SDFs. Considering the particle-based approach, the difference results from the offset of the boundary geometry for the particle-based representation that implicitly narrows small object cavities and holes and blocks

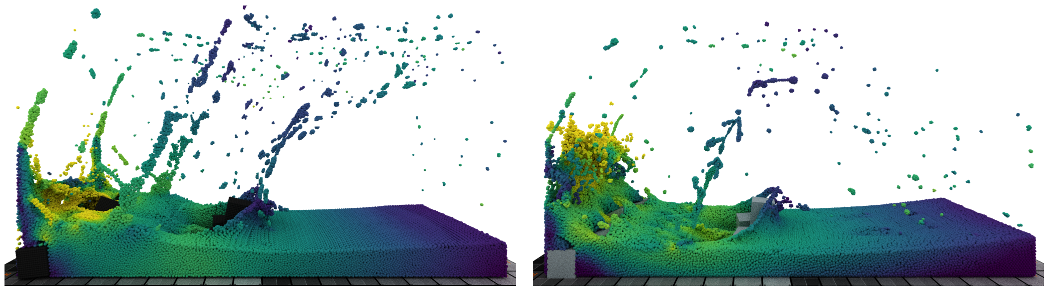

(a) ILSPH [Gis+19] (b) Our approach, uniform resolution

Figure 6.12: Comparison for the *Boxes* scenario, using different boundary handling approaches, where a set of stacked boxes is impacted by a fluid volume. Due to the different representation of the boundaries the results are noticeably different, i.e., the orientation of the box in the bottom left corner as well as the exaggerated splashing due to the representation of the narrow passages between boxes on the initial impact. Velocity color coded from purple to yellow.

fluid particles from entering these regions.

This is in accordance with Gissler et al. [Gis+19] who observed a similar behavior considering rigid-rigid separation. They adjusted the separation distance by tuning the parameter γ in (12) in [Ban+18a], which is not possible for fluid-rigid interactions as this could lead to holes in the boundary geometry; see Fig. 6.6. Fig. 6.12 shows a two-way coupled *Boxes* scene containing two-way coupled boxes being impacted by a fluid volume. We achieve a speedup of $2\times$, as the required iteration count is significantly reduced, which is in accordance with the results for one-way coupling and [Ban+18a]. The difference in boundary representation causes significant differences between the overall flow behaviors; see the results for one-way coupling and Sec. 6.9.2. This difference is due to the feedback between fluid and boundary for two-way coupling, i.e., the fluid's dynamic and the boxes' motions are both influenced by the smoothed boundary-representation in a feedback loop. However, these issues are just an amplification of a problem already discussed in context of Fig. 6.10.

Note that whilst our single contact point model can readily handle two-way fluid-rigid coupling and provides a significant speedup over particle-based methods, simulating strong fluid-rigid coupling, as done by Gissler et al. [Gis+19], is not possible as there is no way to evaluate boundary-boundary interactions in our model and, thus, have to rely on external rigid-rigid solvers, i.e., Bullet [Cou15].

6.9.7 Sampling

Fig. 6.14 compares different fluid particle resolutions and different boundary representations in a simple scenario where fluid is dropped into a basin. We use two different fluid particle resolutions, $r = 0.5$ and $r = 0.075$, and a fixed voxel resolution of 0.5 for the SDF. For the resolution of $r = 0.5$ and $r = 0.075$, the boundary particle sampling requires about 15 seconds and 81K boundary particles and almost fifteen minutes and 2M boundary particles, respectively. At matching fluid and boundary particle resolutions the simulation is stable, i.e., no particles explode in the simulation, but significant problems are visible where particles get stuck on the surface due to the uneven sampling caused by particles, also observed by

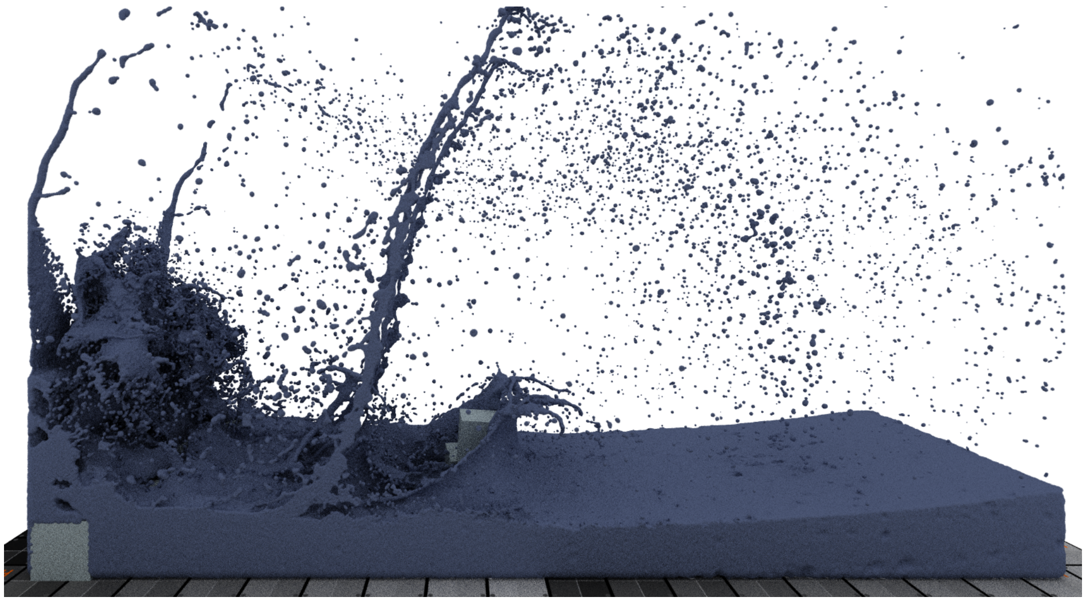


Figure 6.13: Spatially adaptive variant of Fig. 6.12 rendered with an extracted surface using [ZB05]. Adaptivity causes changing resolutions on the outside walls, which causes artifacts during surface extraction.

Band et al. [BGT17]. This sampling introduces noticeable artificial viscosity, which causes the overall fluid behavior to change. In Fig. 6.10 the fluid front for the particle-based approach lages behind the SDF-based boundary with equal fluid resolution. Mismatching resolutions either result in unstable simulations, in case boundary particles are too small, and penetration in case of boundary particles too large; see also Sec. 6.9.2. The boundary-integral-based nature of our approach intrinsically solves this sampling problem and yields stable simulations, as the SDF results in a smooth and continuous boundary representation that is independent from the fluid particle resolution. Fig. 6.15 nicely demonstrates the flexibility of SDFs for very large scenes with small boundary features.

6.9.8 Adaptivity

In this section we briefly summarize the behavior of our boundary handling method related to spatially adaptive fluids. We observe no instabilities directly caused by the interaction of particles of varying resolutions with the same boundary geometry. Moreover, we observe similar flows in adaptive and non-adaptive flows close to boundaries, e.g., Fig 6.12 and Fig. 6.13, due the underlying consistent behavior of our method across varying resolutions; see Sec. 6.9.2. Fig. 6.16 demonstrates this for smoothly varying resolutions along the boundary of a one-way coupled scenario. Regarding computational cost, our boundary handling approach has no significant drawback for an adaptive simulation, i.e., it does not require sampling the boundary with boundary particles of the finest possible fluid particle resolution. However, note that we cannot evaluate the computational cost for a baseline resolution, i.e., all particles are at the finest resolution of the adaptive simulation, as this would require, for example, 133 million particles in the *Dragon* scene. This is significantly above the maximum number of 23 million particles that we can simulate on our hardware, using the data structure approach of Winchenbach and

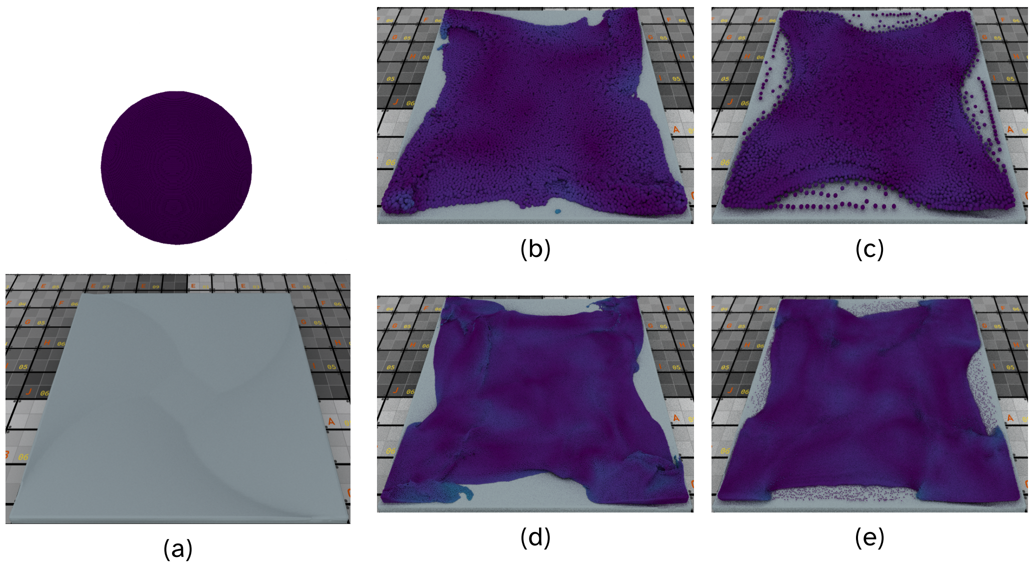


Figure 6.14: Comparison of our semi-analytical boundary handling approach (middle) and a particle-based boundary handling approach (right) considering the behavior of a boundary at a fixed sampling with low (top) and high (bottom) resolution particles. The volume ratio between the low and high fluid particle resolution was about $300 : 1$ with a boundary particle resolution equal to the fluid particle resolution. (a) shows the initial setup of fluid before dropping. Velocity of particles is color coded, with boundary particles being colored dark grey.

Kolb [WK19].

Fast moving objects, i.e., in the *Propeller* scene in Fig. 6.1, require a significantly reduced timestep, but this is due to the requirements imposed by the CFL condition and not specifically due to our boundary handling scheme. Furthermore, for the adaptive method of Winchenbach et al. [WHK17] the particle resolution depends on the distance of a particle to the fluid surface. Existing surface detection methods can sometimes yield incorrect results, i.e., for thin fluid sheets or lone clusters of particles causing particles to needlessly change resolution, which can introduce visual artifacts, that are amplified by parameter scaling problems in surface extraction methods.

6.9.9 Friction

In order to evaluate the friction model presented in Sec. 6.8, we drop a sphere of liquid onto an inclined plane with a high friction coefficient in one case and zero friction in the other case; see Fig. 6.17. This demonstrates the capability of our method to simulate boundaries ranging from free-slip to no-slip boundary conditions. Even though the Coulomb friction model should, from a physical perspective, not be applied to fluid-solid interactions, it still yields visually pleasing results. It has been demonstrated in previous work that a particle-based boundary representation leads to undesirable results due to sampling irregularities [KB17].

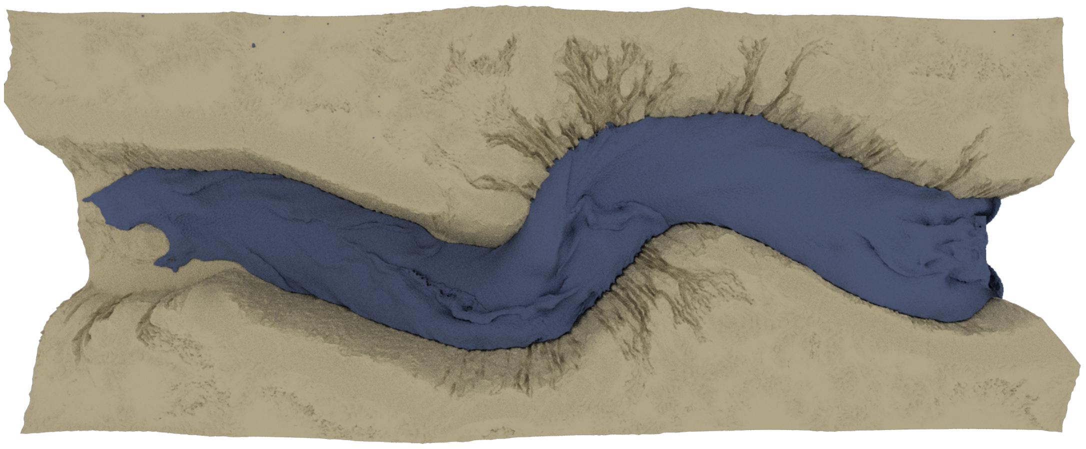


Figure 6.15: This figure shows a *River* scene where a fluid inlet stream, at a particle radius of $0.5m$ moves through a canyon that is 1km long, 400m wide and 200m high. In this scene up to 8 million particles are simulated. The surface is extracted using the approach of Zhu and Bridson [ZB05].

6.9.10 Stability

The stability of our method is evaluated in a complex one-way coupled simulation, shown in Fig. 6.1. Here an inlet stream from the right (initial velocity of $30m/s$) collides with a counter clockwise rotating propeller in a closed housing. Due to the limited space between the housing and the propeller, and due to the propeller pushing the fluid back towards the inlet stream, a significant compressive stress is generated. For a uniform particle resolution, as given in Fig. 6.1, the average timestep requires 42 iterations of the incompressible solver at a CFL limited timestep of 3.4ms. However, the simulation maintains an average compression error of less than 0.01%. In general, high compressive stresses are in regions of low resolution in the fluid bulk, and not at the free surface with higher particle resolution. Consequently, when using an adaptive particle resolution in this scenario (Fig. 6.1), the average timestep is reduced to $740\mu s$ as the smallest particle has a 4.6 times smaller radius. This, in turn, reduces the timestep by a factor of 4.6 due to the CFL condition. As a consequence, the average number of iterations of the pressure solver is reduced to 3, which significantly reduces the computational requirements per particle. All in all, 22 timesteps are required per rendered frame, which significantly slowed down the overall simulation.

To further evaluate the stability of our method, we simulate a fluid stream flowing past a cylinder through a tube in 3D; see Fig. 6.18. For this simulation we include the background pressure term of Marrone et al. [Mar+13], to avoid the formation of voids behind the cylinder. Our boundary handling approach is able to simulate this scenario, including vortex shedding behind the cylinder, even for inviscid fluids at high velocities. Due to the smooth nature of the boundaries in this scene our method closely approximates the results of a boundary-integral approach.

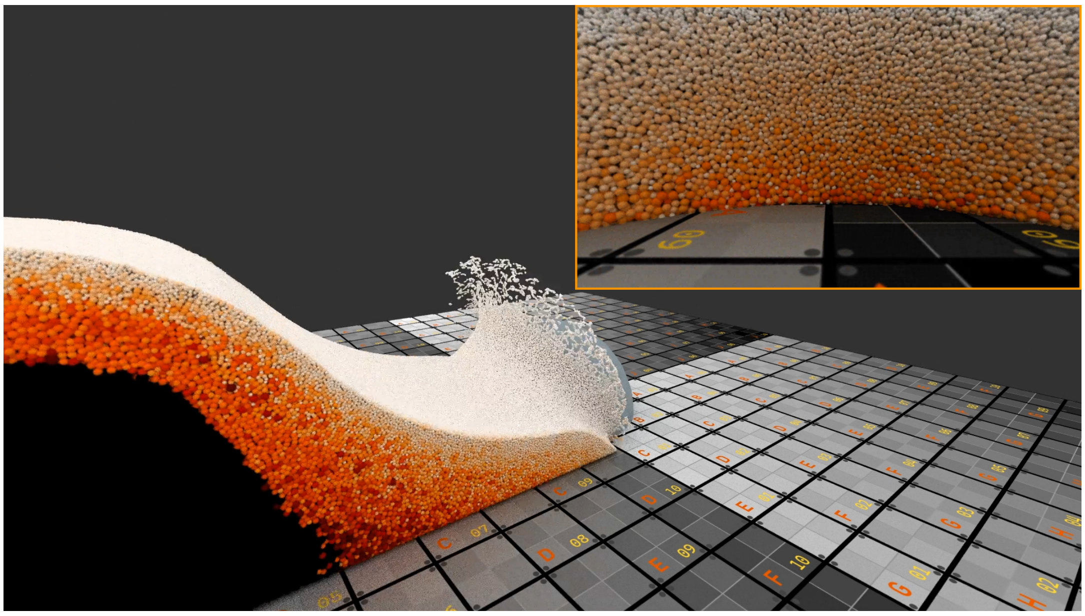


Figure 6.16: An adaptive dambreak simulation where an initial fluid volume impacts a smooth hemisphere. The top right close-up shows a view from inside the hemisphere to demonstrate the changing resolution along the boundary. Particle volume color coded.

6.9.11 Limitations

Our method has three major limitations. Firstly, SDFs can not represent all types of geometries consistently. In regions where multiple parts of a single rigid body are within the support radius of a particle, e.g., in narrow cavities, the single contact point model might not accurately represent the interaction. However, this effect can be reduced by performing a convex body decomposition, apriori, yielding multiple contact points. Secondly, single particles get trapped in sharp boundary features. Even though this does not negatively impact the flow behavior, this is an undesired property which we intend to address using improved penalty functions in future work. Finally, our approach cannot readily handle deformable boundary geometries, without significant computational overhead. Additionally, certain aspects of SPH, in general, are not yet well applied to spatially adaptive methods, i.e., surface-detection and surface-extraction methods, inducing visual artifacts. Moreover, certain aspects of spatially adaptive SPH methods, i.e., how to merge large amounts of high-resolution particles, are challenging problems, in general, and can potentially cause simulations to not behave well.

6.10 Conclusions

In this paper we presented a novel semi-analytic boundary handling approach for SPH simulations. The core idea is to define a locally representative planar boundary, for arbitrary boundary geometries, using an SDF, and to derive an analytic solution for the interaction with planar boundaries to this representative boundary. Our method yields significantly improved preservation of detail for small boundary features and provides consistent boundary interactions across varying particle

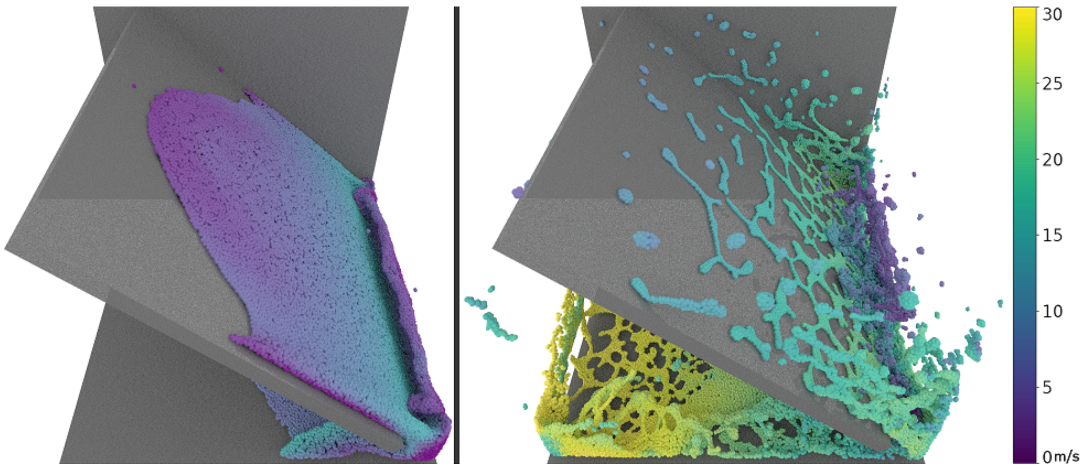


Figure 6.17: Simulation of the collision of an initial spherical fluid volume falling on an inclined surface. On the left we set the friction coefficient μ for all boundaries equal to 1, whereas the right uses a friction coefficient $\mu = 0$. Our method is readily capable of simulating a wide range of fluid-rigid interactions, ranging from free-slip (right) to no-slip boundary conditions. Velocity color coded from purple ($0m/s$) to yellow ($30m/s$).

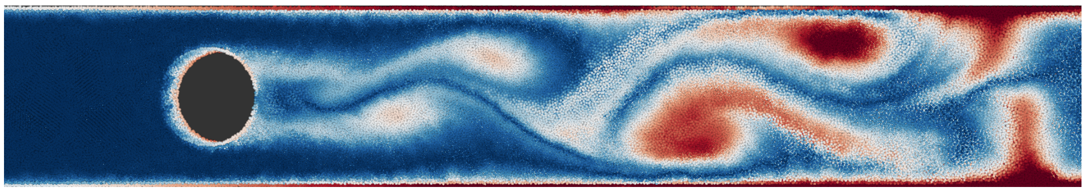


Figure 6.18: This figure demonstrates a flow past a cylinder simulation in 3D for an inviscid fluid demonstrating vortex shedding behind the obstacle. Local angular velocity is color coded from zero (blue) to high (red). Note that particles close to the boundary have a relatively high angular velocity as the boundary is not moving relative to them.

resolutions. Additionally, our approach can readily be integrated into existing SPH simulation methods and can handle spatially adaptive simulations, even in two-way coupled scenarios.

6.A Appendix

The analytical solution in 2D required solving a set of definite integrals:

$$\lambda_2(d) = \begin{cases} \int_d^{0.5} f(q,d)dq + \int_{0.5}^1 g(q,d)dq, & 0 \leq d \leq 0.5 \\ \int_d^1 g(q,d)dq, & 0.5 < d \leq 1, \\ 1 - \lambda_2(-d), & -1 \leq d < 0 \end{cases}$$

$$f(q,d) = 2\,\text{acos}\left(\frac{d}{q}\right) qC_2 \left[(1-q)^3 - 4\left(\frac{1}{2} - q\right)^3\right], \tag{6.56}$$

$$g(q,d) = 2\,\text{acos}\left(\frac{d}{q}\right) qC_2 \,(1-q)^3\,,$$

which can be evaluated symbolically. However, the resulting intermediate definite integrals are skipped for brevity. The final boundary contribution term $\lambda_2(d)$ in 2D can be determined using the definite integrals from before as

$$\lambda_2(d) = -\frac{1}{7\pi}\begin{cases} d_1 + d_2 + d_3 + d_4 + d_5 + d_6, & 0 \le d < 0.5 \\ e_1 + e_2 + e_3 + e_4, & 0.5 \le d \le 1 \\ 7\pi\mathcal{W}(-d) - 7\pi, & -1 \le d < 0, \end{cases} \tag{6.57}$$

with

$$\begin{aligned} d_1 &= \left(-12d^5 - 80d^3\right)\log\left(2\sqrt{1-d^2}+2\right) \\ d_2 &= \left(30d^5 + 80d^3\right)\log\left(\sqrt{1-4d^2}+1\right) \\ d_3 &= -18d^5\log\left(1-\sqrt{1-4d^2}\right); d_4 = \operatorname{acos}(2d) - 8\operatorname{acos}(d) \\ d_5 &= \sqrt{1-d^2}\,\left(68d^3+32d\right); d_6 = \sqrt{1-4d^2}\,\left(-68d^3-8d\right) \\ e_1 &= \left(-6d^5-40d^3\right)\log\left(\sqrt{1-d^2}+1\right); e_2 = -8\operatorname{acos}(d) \\ e_3 &= \left(6d^5+40d^3\right)\log\left(1-\sqrt{1-d^2}\right); e_4 = \sqrt{1-d^2}\,\left(68d^3+32d\right) \end{aligned} \tag{6.58}$$

The analytical solution in 3D required solving a set of definite integrals

$$\begin{aligned} \lambda_3(d) &= \begin{cases} \int_d^{0.5} f(q,d)dq + \int_{0.5}^1 g(q,d)dq, & 0 \le d \le 0.5 \\ \int_d^1 g(q,d)dq, & 0.5 < d \le 1, \\ 1 - \lambda_3(-d), & -1 \le d < 0, \end{cases} \\ f(q,d) &= C_3\left[(1-q)^3 - 4\left(\frac{1}{2}-q\right)^3\right]2\pi q(q-d), \\ g(q,d) &= C_3\,(1-q)^3\,2\pi q(q-d), \end{aligned} \tag{6.59}$$

which result in

$$\begin{aligned} \int_d^{0.5} f(q,d)dq &= \frac{192d^6 - 288d^5 + 160d^3 - 66d + 19}{60}, \\ \int_{0.5}^1 f(q,d)dq &= -\frac{18d-11}{60}, \\ \int_d^1 g(q,d)dq &= -\frac{16d^6 - 72d^5 + 120d^4 - 80d^3 + 24d - 8}{15}. \end{aligned} \tag{6.60}$$

In order to calculate $\mathcal{S}$ in 3D the following definite integrals

$$\begin{aligned} \mathcal{S}_3(d) &= \begin{cases} \int_d^{0.5} f(q,d)dq + \int_{0.5}^1 g(q,d)dq, & 0 \le d \le 0.5 \\ \int_d^1 g(q,d)dq, & 0.5 < d \le 1, \\ \int_0^{0.5} f(q,d)dq + \int_{0.5}^1 g(q,d) - \mathcal{S}_3(-d), & -1 \le d < 0, \end{cases} \\ f(q,d) &= \frac{4608}{\pi}\left[q^3\,(q-d)\,(3q-2)^2\right], \\ g(q,d) &= \frac{4608}{\pi}\left[q\,(q-d)\,(q-1)^4\right], \end{aligned} \tag{6.61}$$

were solved as

$$\begin{aligned}
&\int_0^{0.5} f(q,d)dq = \frac{f_1+f_2}{420\pi}, \int_d^{0.5} f(q,d)dq = -\frac{2337d-578}{420\pi},\\
&\int_{0.5}^1 g(q,d)dq = -\frac{81d-46}{252\pi}, \int_d^1 g(q,d)dq = \frac{g_1+g_2}{63\pi},\\
&f_1 = 26880d^9 - 69120d^8 + 46080d^7 + 21504d^6,\\
&f_2 = -32256d^5 + 8960d^3 - 2337d + 578,\\
&g_1 = 448d^9 - 3456d^8 + 11520d^7 - 21504d^6,\\
&g_2 = 24192d^5 - 16128d^4 + 5376d^3 - 576d + 128,
\end{aligned} \tag{6.62}$$

which results in a final boundary term in 3D as

$$\mathcal{S}_3(d) = \begin{cases} \frac{1}{h^{d+2}}\frac{1}{315\pi}\left[s_1+s_2\right], & 0 \le d \le 0.5\\ \frac{1}{h^{d+2}}\frac{1}{63\pi}\left[t_1+t_2\right], & 0.5 < d \le 1,\\ -\frac{1}{630\pi}\left[5913d-5392\right] - \mathcal{S}(-d), & -1 \le d < 0. \end{cases} \tag{6.63}$$

with

$$\begin{aligned}
s_1 &= 20160d^9 - 51840d^8 + 34560d^7,\\
s_2 &= +16128d^6 - 24192d^5 + 6720d^3 - 1854d + 491,\\
t_1 &= 448d^9 - 3456d^8 + 11520d^7,\\
t_2 &= -21504d^6 + 24192d^5 - 16128d^4 + 5376d^3 - 576d + 128
\end{aligned} \tag{6.64}$$

The naïve gradient term, as shown in Sec. 6.5.4, can also be found for particle-based terms, i.e., [Pri12]

$$\nabla_i A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla_i W_{ij}, \tag{6.65}$$

which also is not exact for constant functions and also does not preserve symmetric interactions. For a particle-based representation a term that is exact for constant functions [Pri12] can be found for boundary integral approaches as

$$\nabla A_i = \sum_j \frac{m_j}{\rho_j}(A_i - A_j)\nabla_i W_{ij} \rightarrow \frac{\rho_b^0}{\rho_b}\left(A_i - A_B\right)\nabla_i \lambda_n\left(d\right), \tag{6.66}$$

and one that preserves symmetric interactions as

$$\nabla A_i = \rho_i \sum_j m_j \left(\frac{A_i}{\rho_i^2} - \frac{A_j}{\rho_j^2}\right)\nabla_i W_{ij} \rightarrow \rho_i \rho_b^0 \left(\frac{A_i}{\rho_i^2} - \frac{A_B}{\rho_b^2}\right)\nabla_i \lambda_n\left(d\right), \tag{6.67}$$

utilizing the same derivation as Price [Pri12].

Chapter 7

Optimized Refinement for Spatially Adaptive SPH

Contextualization

This chapter reprints the paper "Optimized Refinement for Spatially Adaptive SPH"[1] originally published in ACM Transactions on Graphics [WK21] and presented at SIGGRAPH 2021. In this paper an adaptive SPH model is described that builds on prior work, see Chapter 3, and that improves many of the limitations of this prior work. The most notable improvement is with regards to refinement patterns that were constructed based on intuition and manual tweaking in prior work, which made them both not readily reproducible and not physically motivated. Furthermore, this paper also introduces a novel concept of local viscosity to reduce the impact of errors during refinement which significantly helps in making adaptive SPH methods more applicable to practical scenarios.

The core idea of the paper is a combination of a priori and online optimization steps that adjust refinement patterns a priori for idealized isotropic particle distributions and online during a simulation to actual particle neighborhoods. By combining these optimized refinement patterns, a modified implicit blending process and a local viscosity model, the overall adaptive simulation becomes significantly more stable and requires lower damping through artificial viscosity effects. Furthermore, the paper highlights fundamental limits to the residual error in particle refinement and, accordingly, motivates the usage of error-damping approaches, e.g., blending and local viscosity, to reduce the impact of these residual errors.

The initial motivation of the optimization came out of the prior work on adaptive SPH methods, see Chapter 3, where manually tuned optimization patterns were utilized that were extremely hard to reproduce. Based on Vacondio et al. [Vac+13], Rene Winchenbach developed the initial concept of the optimization process, and he deduced all conceptual details and implementational aspects. Andreas Kolb provided valuable feedback, especially with regards to the difficult derivation of the method and supported its presentation, as well as the writing of the final paper. Special thanks is also owed to the anonymous reviewers of the initial submission of the paper to ACM Transactions on Graphics, as they provided extensive feedback that helped in significantly improving the paper in a major revision.

[1]Note that this paper denotes adaptivity ratios as $1 : n$, i.e., 1 low resolution particle is equivalent to n high resolution particles, whereas most other papers denote adaptivity ratios as $n : 1$.

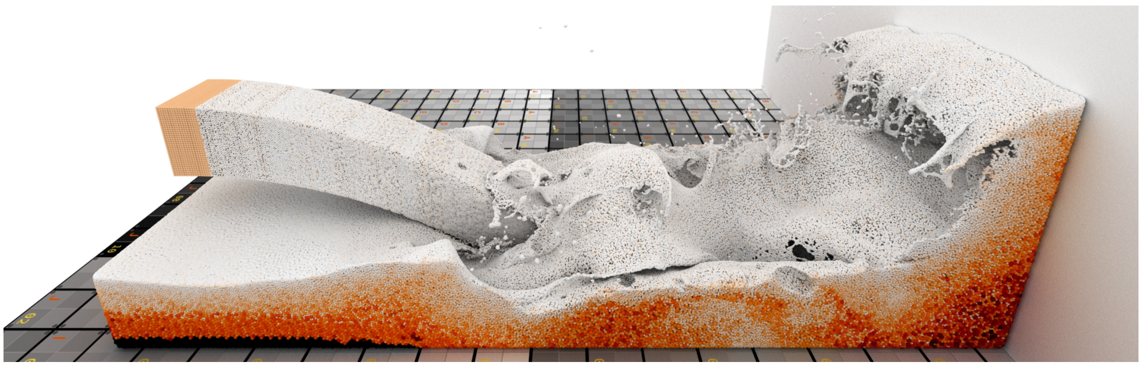


Figure 7.1: An inlet, emitted with a moderate resolution, collides with a fluid volume that is agitated by a moving boundary wall. Our adaptive method can easily handle the $1:500$ adaptive volume ratio shown here, without instabilities, and can readily adapt the resolution of the inlet on the fly without causing the inlet to be decollimated. Volume color mapped from high (black) to low (white).

Abstract

In this paper we propose an improved refinement process for the simulation of incompressible low-viscosity turbulent flows using Smoothed Particle Hydrodynamics, under adaptive volume ratios of up to $1:1,000,000$. We derive a discretized objective function, which allows us to generate ideal refinement patterns for any kernel function and any number of particles a priori without requiring intuitive initial user-input. We also demonstrate how this objective function can be optimized online to further improve the refinement process during simulations by utilizing a gradient descent and a modified evolutionary optimization. Our investigation reveals an inherent residual refinement error term, which we smooth out using improved and novel methods. Our improved adaptive method is able to simulate adaptive volume ratios of $1:1,000,000$ and higher, even under highly turbulent flows, only being limited by memory consumption. In general, we achieve more than an order of magnitude greater adaptive volume ratios than prior work.

7.1 Introduction

Fluid simulations play an important role in modern computer animations, and there is an ever-increasing demand not just for more vivid and higher quality free surface fluid simulations, but also larger overall simulation domains. However, a uniform resolution in all parts of the simulation is not ideal as this requires high resolutions in regions of the fluid where the behavior is not interesting, e.g.,at the bottom of a pool. Therefore, methods are needed that can focus the computational resources to where it is most beneficial. This can be achieved using spatially adaptive methods, which have been used for grid-based simulations for a while, e.g., using octrees [LGF04], tetrahedral meshes [Kli+06], or tetrahedral meshes combined with an adaptive FLIP simulation [ATW13]. For grid-free methods, based commonly on Smoothed Particle Hydrodynamics (SPH), prior research mainly focused on weakly compressible simulations [Ada+07; FB07] and only recent work [WHK17] has enabled incompressible flows with larger refinement ratios.

An adaptive SPH method starts by defining a desired resolution, often based on surface distance, for each particle using a sizing function. The resolution is then locally decreased by merging particles, smoothed using sharing between particles, or increased by refining a particle into multiple smaller particles. These processes can introduce errors into the simulation and in order to stabilize the simulation blending methods have been proposed [OK12; WHK17]. Most of the introduced refinement error is due to a reliance on some intuition [FB07; Vac+13; WHK17] instead of a fundamental analytical model for particle refinement patterns.

In order to improve the refinement process, we first introduce a continuous objective function describing these processes, in Sec. 7.5, which can be applied to symmetric SPH formulations required in adaptive incompressible flows. Sec. 7.6 then demonstrates our novel discretization technique based solely on particles and derives the error terms, as well as their derivatives, required for efficient optimization. In Sec. 7.7 we then utilize our discretization technique to optimize refinement patterns regarding positions and mass distributions, both a priori for ideal environments and online using actual particle neighborhoods, using these discrete error terms. Our results show an unavoidable inherent residual refinement error, which we smooth out using improved and novel techniques to ensure stability, in Sec. 7.8. Finally we demonstrate the efficacy of our improved method and its properties in Sec. 7.10.1 by comparing it to prior work, and identifying possible factors that could further improve stability.

7.2 Related work

In the past, spatially adaptive SPH methods have been widely used within the CFD context, e.g.,by Vacondio et al. [VRS12], Feldman and Bonet [FB07] and Li et al. [LWQ15], and to some extent within computer animation, e.g.,by Solenthaler and Gross [SG11], Orthmann and Kolb [OK12], Horvath and Solenthaler [HS13], and more recently by Winchenbach et al. [WHK17]. These adaptive methods use different approaches to resolve underlying stability issues, e.g.,using multiple separate simulations, temporal blending methods or, as done in most of these methods, by using a weakly compressible SPH formulation. Older methods, e.g.,Adams et al. [Ada+07], developed before modern incompressible pressure solvers existed (starting with PCISPH [SP09]), did not have to consider as strict stability requirements. However, all these adaptive methods need particle patterns for replacing a particle of lower resolution by particles of higher resolution. This replacement, however, introduces a density error that needs to be addressed to avoid instabilities.

Solenthaler and Gross [SG11] address the stability issue by utilizing separate simulations, each using a different global uniform level of resolution, where particles are directly inserted and those causing large errors are simply removed. Orthmann and Kolb [OK12] apply a simple 1:2 splitting pattern and a temporal blending scheme, which significantly limits the temporal rate of resolution change possible. Vacondio et al. [Vac+16] propose statically optimized patterns to avoid the density errors but require asymmetric support radii, which are not stable for incompressible SPH methods. Winchenbach et al. [WHK17] combine manually optimized refinement patterns with temporal blending to allow for adaptive incompressible

SPH simulations. However, this approach does not fully solve the instability problem and strongly depends on manual parameter tuning.

Various approaches have recently been developed for SPH-based simulations in computer animation, e.g.,implicit incompressible SPH (IISPH) [Ihm+13], divergence-free SPH (DFSPH) [BK15] and position based fluids [MM13]. These approaches predict and correct fluid compression through predictions in each time step, but are not designed to handle sudden fluid compressions, such as those caused by particle splitting. Therefore, errors due to particle refinement need to be prevented from occurring in the first place by changing the adaptive method itself, instead of relying on an external pressure solver.

Furthermore, rigid boundary handling is an existing issue for adaptive methods [WHK17], as commonly used particle boundaries [Aki+12] suffer from sampling problems. Fujisawa and Miura [FM15] address the sampling problem by using an analytical integral formulation of boundaries. Koschier and Bender [KB17] extended this approach using numerical integrals precomputed on grids. Recently, Winchenbach et al. [WAK20] introduced a signed distance field based boundary integral method that does not require expensive precomputation steps and ensures consistent behavior across varying particle resolutions. Band et al. [BGT17] utilize a moving least squares method to fit planes to particles for sufficiently flat boundaries. The method is then used to calculate accurate pressure values for boundary particles instead [Ban+18b]. Finally, Gissler et al. [Gis+19] introduced an extended SPH model to simulate rigid to rigid interactions using SPH. Boundary integral based methods are a good general choice, as they avoid the sampling issue, but also introduce non-SPH representations.

Finally, temporally adaptive methods are alternative approaches to allocate computational resources where they are most beneficial. Adjusting the time step of an SPH simulation, globally, using the CFL condition has found wide adoption in SPH, with initial work by Desbrun and Gascuel [DG96] and later by Ihmsen et al. [Ihm+10]. Assigning different particles, or regions, different time steps has also been proposed by Desbrun and Gascuel [DG96], however, this approach has not find wide adoption, due to the complexity involved in synchronizing different time steps. Reinhardt et al. [Rei+17] applied this concept, through regional-time-stepping, to modern pressure solvers, but is only applicable to CPU-based SPH variants.

7.3 Foundations of SPH

SPH is a numerical method to solve continuous integrals by approximating an underlying continuous system using discrete particles, see [Mon92] and for further explanations [Kos+19]. These particles are then used to approximate continuous quantities using the basic SPH interpolation operator for a particle i utilizing all neighboring particles j, which is given by

$$A(\mathbf{x_i}) = \sum_j \frac{m_j}{\rho_j} A(\mathbf{x_j}) W_{ij}, \tag{7.1}$$

where m, ρ and $\mathbf{x}_i$ denote the mass, density and position respectively, and the subscript denotes indices of the particle. $W_{ij} = \hat{W}(|\mathbf{x}_{ij}|, h_{ij})$ is a radially symmetric kernel function that depends on the distance between positions $|\mathbf{x}_{ij}| = |\mathbf{x}_i - \mathbf{x}_j|$ and the support radius of the interaction h_{ij}. This term can be chosen as asymmetric, e.g.,as $h_{ij} = h_i$, which results in a gather formulation, or as $h_{ij} = h_j$, which results in a scatter formulation, or as the average support radius, $h_{ij} = \frac{h_i+h_j}{2}$, which results in a symmetric formulation. For adaptive incompressible SPH only a symmetric formulation is stable [WHK17] , whereas for adaptive weakly compressible SPH (WCSPH) a common choice is the scatter formulation as this significantly simplifies many derivative terms [Vac+13].

Commonly used kernel functions include the cubic spline kernel [Mon05] [Ihm+13; KB17] and the Wendland kernel functions [Vac+16]. The exact choice of kernel function does not influence our method, but still changes the support radius of a particle. Every kernel function has an ideal number of neighbors [DA12], i.e., $N_H = 50$ for the cubic spline kernel, which yields the support radius for a particle by solving $\frac{4}{3}\pi h^3 = N_H V_i$, as there should be N_H particles of volume V_i contained within a spherical support radius H. The support radius H is related to the smoothing scale h through a factor $\frac{H}{h}$. Within Computer Animation, $\frac{H}{h}$ is often defined as 1 [Kos+19] and, thus h is also often referred to as support radius. We will follow this notion in our paper.

7.4 Method overview

Spatial adaptive methods generally begin by determining the desired resolution for a particle using a sizing-function, e.g.,using the particle's surface distance, or based on visibility, and classifying particles accordingly into different categories [WHK17] and then adjusts the resolution of each particle to be closer to its desired resolution. This can, in general, be done using three distinct processes:

Merging: This process combines multiple high-resolution particles into lower resolution particle(s) to reduce the local spatial resolution. This can be done in multiple ways, e.g., merging 2 particles into 1 ($2:1$-merging), distributing one particle onto several other particles ($n:n-1$-merging) or combining many particles into one ($n:1$-merging). This process is fully constrained for $n:1$ merging due to mass and momentum conservation.

Sharing: This process changes a particle that is larger than its ideal resolution by distributing parts of its mass and quantities to nearby particles which are smaller than their ideal resolution. This process is essentially an extension of merging, where the original particle remains in the simulation. The process for interactions between two particles is also fully constrained, due to mass and momentum conservation.

Splitting: This process splits (or refines) a particle that is significantly larger than its ideal resolution into multiple smaller particles. Quantities of the inserted particles are not fully constrained, i.e.,the mass of each created particle can vary as long as the overall mass is preserved. Additionally, geometric patterns have been used in all prior methods to insert new particles into the simulation, but these often require significant manual tuning to achieve a certain level of stability. In general, different refinement methods rely on different splitting steps, e.g., $1:2$ or $1:8$. However some methods allow for arbitrary changes (up to a certain limit) of $1:n$. Some splitting methods include procedures to reduce the impact of errors

ALGORITHM 7.1: An overview of the off-line a priori and the online optimization process to generate the initial refinement pattern and its local adaptation during simulation, respectively.

```
// A priori optimization of $n$ patterns
For all refinement patterns
    Initialize pattern with random positions and uniform weights
    Optimize positions; see Sec. 7.6.1 and 7.7.1
    Optimize weight distribution; see Sec. 7.6.2 and 7.7.1
    Optimize positions and weights simultaneously 7.7.1
    Store pattern for refinement

// Online optimization of particle $i$
Determine ideal particle radius $s_i$; see Sec. 7.9
If particle radius $r_i \leq 2s_i$: return // no refinement
Determine pattern to be used $p = \lfloor clamp(r_i/s_i, 2, n) \rfloor$
Initialize refined particles using a priori pattern for $p$ particles
Optimize positions using gradient descent; see Sec. 7.6.1 and 7.7.2
Optimize masses using evolutionary optimization; see Sec. 7.7.2
Initialize blend process for refined particles; see Sec. 7.8.1
Insert refined particles and remove old particle
```

introduced by the refinement process.

The merging and sharing processes are mostly limited by the search for eligible particle groups, often resulting in $2:1$ merging and $1:1$ sharing being used. These processes are fully-constrained by mass and momentum conservation, and also do not cause visually apparent instabilities. The splitting process, however, causes instabilities, i.e., due to changes to the density field, and can be optimized [FB07]. Refining a single low resolution particle o, with mass m_o at position $\mathbf{x}_o$ and velocity $\mathbf{v}_o$, into n particles, i.e.,a $1:n$ split, has to preserve mass, kinetic energy, as well as linear and angular momentum, and should not modify the underlying density field to avoid compression. Mass-conservation can be enforced by ensuring that the total mass of the refined particles is equal to m_o, whereas momentum and kinetic energy are conserved if and only if the velocities of the refined particle are equal to $\mathbf{v}_o$ [Fel06]. Consequently, the error introduced on the underlying density field can be controlled by optimizing the mass distribution and positions for the refined particles. This yields $4n$ degrees of freedom, making manual parameter tuning impractical.

The approach, to optimize the refinement, we present is independent of the number of high-resolution particles being created and of the kernel function used, and is applicable to any adaptive method that relies on particle refinement and any incompressible or weakly compressible solver. However, we first require an underlying objective function for a symmetric SPH formulation, as prior optimization approaches relied on an asymmetric scatter-based SPH formulation. Our overall refinement process is described in Algorithm 7.1.

7.5 Continuous objective functions

In general, the splitting process works by replacing the original low resolution particle o with a set of higher resolution particles $\mathcal{S}$. The problem now is based on the choice of parameters for the new particles, where for n particles we get $d \cdot n$ positional parameters (for d dimensions) and n weights determining the mass distribution. The distributed mass needs to equal the original mass, e.g.,$m_o = \sum_{s\in\mathcal{S}} m_s$, due to mass conservation. However, there is no way to directly determine the patterns and prior work often employed fixed refinement patterns, e.g., 1:2 [OK12], 1:7 [SG11], or 1:13 [Vac+16] or for multiple n [WHK17], which are based on using intuitive shapes and often involve manual tuning.

Mass and momentum are directly conserved. The underlying fields that should be kept constant are the density and velocity fields [Fel06], with a focus on the density field, as a change to density would introduce significant instabilities. The change to the density field at any point $\mathbf{x}$ can be defined, based on the density before $\rho(\mathbf{x})$ and after $\rho^*(\mathbf{x})$ refinement [Fel06]

(7.2) $$\tau(\mathbf{x}) = \rho^*(\mathbf{x}) - \rho(\mathbf{x}),$$

which can be evaluated over a continuous domain Ω as

(7.3) $$\mathcal{E} = \int_\Omega \tau(\mathbf{x})^2 d\mathbf{x}.$$

This can be seen as a continuous objective function, where the ideal refinement process would cause a total error of 0. This can also be seen as enforcing density invariant refinement, i.e.,$\frac{D\rho}{Dt} = 0$ [BK15], but instead of a change over time the change occurs due to particle refinement. Feldman and Bonet [FB07] use an a priori optimization process to minimize $\mathcal{E}$, using an initial refinement pattern found by intuition, i.e.,an icosahedra in 3D configuration. Fixing the relative positions of particles reduces the problem to determining a single scaling parameter and the mass distribution, which Feldmann and Bonet solve for a scatter-based formulation of SPH. However, the approach of Feldman and Bonet [FB07] is not directly applicable to incompressible SPH simulations, as these require a symmetric SPH formulation to avoid instabilities [WHK17].

For a symmetric SPH formulation, the support radius of an interaction depends on the definition of a support radius $h(\mathbf{x})$ for every integration point, which could be determined using the approaches from Monaghan [Mon02] and Winchenbach et al. [WHK17] that base the support-radius on the density at this position, which also means that particles of equal resolution have varying support-radii, i.e., $V_i = V_j \not\Rightarrow h_i = h_j$. These approaches, however, introduce a coupled problem (refer to [Pri12] for a more thorough examination), where the evaluation of the density depends on the support radius which, again depends on the density. This problem can be stated as

(7.4) $$\rho_i = \sum_j m_j W\left(|\mathbf{x}_i - \mathbf{x}_j|, \frac{h_i + h_j}{2}\right), \quad h_i = \eta\sqrt[3]{\frac{m_i}{\rho_i}},$$

where η is a parameter used to determine the number of neighbors for a particle, commonly chosen as $\eta \approx 2.6$ for cubic spline kernels [WK19]. This term could be

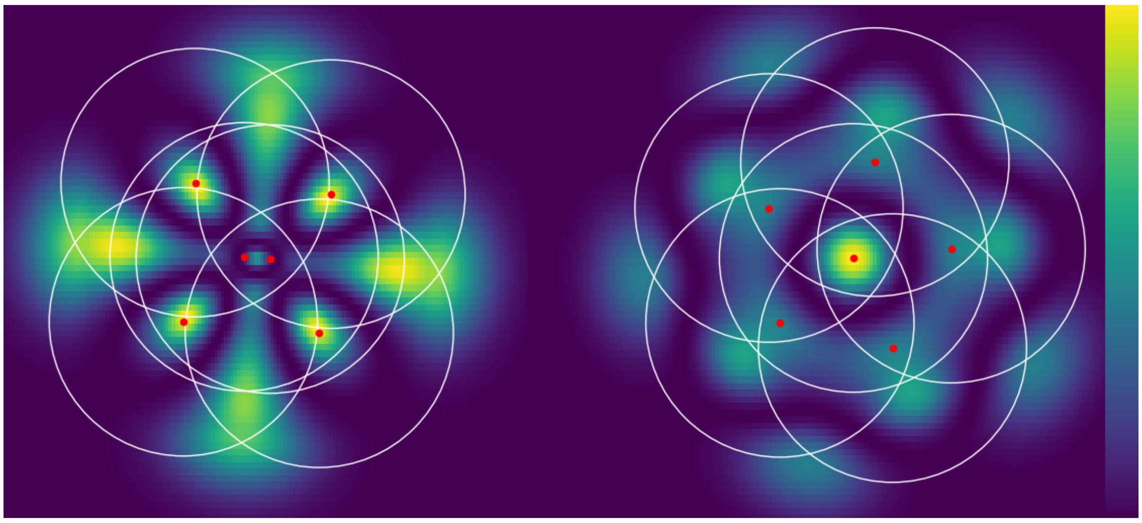


Figure 7.2: The result of optimizing the refined particle positions (red) for a symmetric SPH formulation (left) and a scatter SPH formulation (right), with the density field error $\tau(\mathbf{x})$ color coded from purple to yellow.The error is visualized from 0 to $6 \cdot 10^{-4}$ for symmetric SPH on and the left from 0 to $1 \cdot 10^{-4}$ for asymmetric SPH on the right. Note that the errors in the converse terms, e.g., the right pattern evaluated utilizing the symmetric error metric, result in orders of magnitude worse behavior, highlighting the need to chose the appropriate formulation for the optimization process.

evaluated iteratively until the result converges, but this would require a very expensive computation for every point of integration. Nonetheless, the symmetric formulation can still be used to optimize the refined positions $\mathbf{x}_s$ for all refined particles $s \in \mathcal{S}$, even though the process is significantly slower than for a scatter formulation. The results of this optimization are shown in Fig. 7.2, where the generated distribution of particles is significantly different between the symmetric and scatter formulation, i.e., two particles are placed close to the position of the original particle instead of one. The patterns $\mathcal{P}_s$ and $\mathcal{P}_a$ are generated using a symmetric ($\mathcal{E}_s$) and asymmetric ($\mathcal{E}_a$) formulation of Eqn. 7.3, respectively. Here, $\mathcal{E}_s(\mathcal{P}_s) = 2.4 \cdot 10^{-4}$, $\mathcal{E}_a(\mathcal{P}_s) = 5.8 \cdot 10^{-4}$, $\mathcal{E}_s(\mathcal{P}_a) = 3.0 \cdot 10^{-3}$ and $\mathcal{E}_a(\mathcal{P}_a) = 3.8 \cdot 10^{-5}$. However, this does not mean that one of the generated patterns is superior to the other as they are optimized for fundamentally different SPH formulations, i.e., they are not alternative options. As such, the appropriate pattern should be chosen for each formulation and applying patterns optimized for asymmetric SPH formulations leads to a significantly worse error (by an order of magnitude).

7.6 Discretized objective function

Minimizing Eqn. 7.3 by iteratively solving Eqn. 7.4, is not practical, especially in 3D, due to computational costs. Instead, we propose to only evaluate Eqn. 7.2 at the current positions of particles, which significantly reduces the computational complexity as this only requires $N_H + n$ evaluations, for n refined particles. We denote the original particle as o, the set of refined particles of o as $\mathcal{S}_o$, the set of neighbors of o, including o, as $\mathcal{N}_o$ and the discretized error term $\mathbb{E}$ for the surrounding particle positions and newly inserted particle positions as $\mathbb{E}_{\mathcal{N}}$ and $\mathbb{E}_{\mathcal{S}}$, respectively. For clarity we will drop the subscript on $\mathcal{S}$ and $\mathcal{N}$ whenever possible..

Following Vacondio et al. [Vac+13], we can evaluate the error on neighboring particle positions, by effectively removing the old particle and inserting the refined particles, as

$$\tau_n = \sum_{s \in \mathcal{S}} m_s W_{ns} - m_o W_{no}, \forall n \in \mathcal{N}. \tag{7.5}$$

For a refined particle $s \in \mathcal{S}$, we can calculate the error term as the difference between the original particles density ρ_o and the density evaluated at the current position of s, which resembles the numerical SPH approximation error. This results in the following error term τ_s for a specific refined particle s:

$$\tau_s = \left(\sum_{n \in \mathcal{N}} m_n W_{sn} + \sum_{k \in \mathcal{S}} m_k W_{sk} \right) - \rho_o, \forall s \in \mathcal{S}. \tag{7.6}$$

The discretized error terms are then the mass weighted sum of square error terms, per particle, which yields

$$\mathbb{E}_{\mathcal{N}} = \sum_{n \in \mathcal{N}} m_n \tau_n^2, \quad \mathbb{E}_{\mathcal{S}} = \sum_{s \in \mathcal{S}} m_s \tau_s^2. \tag{7.7}$$

The overall minimization problem can then be defined as minimizing the positions and masses of the inserted particles, under the constraint $\sum_{s \in \mathcal{S}} m_s = m_o$ due to mass-conservation, as

$$\min_{\mathbf{x}_{\mathcal{S}}, m_{\mathcal{S}}} \mathbb{E} = \min_{\mathbf{x}_{\mathcal{S}}, m_{\mathcal{S}}} \left(\mathbb{E}_{\mathcal{N}} + \mathbb{E}_{\mathcal{S}} \right). \tag{7.8}$$

The partial derivatives of this minimization problem with respect to positions $\mathbf{x}_s$ and masses m_s are described in Sections 7.6.1 and 7.6.2, respectively.

Note that this problem has a trivial, but useless, solution where all refined particles are placed at the original particles position and a single refined particle having the mass of the original particle, i.e.,all other refined particles have zero weight. We avoid this trivial solution by iteratively optimizing positions and masses and by imposing a limit on the optimized masses. Thus, we add the constraint

$$\nexists m_s \geq 8 m_i, \forall i, s \in \mathcal{S}, \tag{7.9}$$

which restricts the largest ratio of masses between refined particles to be at most 8. These restrictions additionally prevent degenerate optimization solutions where single particles have zero mass.

This problem can be solved in ideal, isotropic and hexagonal, particle distributions using initially random positions and mass ratios for the refined particles a priori. Due to the relatively low computational cost, it can also be solved online for an actual particle in a fluid simulation using the results of the a priori optimization as a starting point. Fig. 7.3 shows the result of optimizing the particle distribution for 6 particles using our proposed discretization (purple) and by using the continuous form (red), where the results are very similar (ignoring rotation and translation), demonstrating a close approximation of the underlying problem. The minimization problem is derived in detail in the appendix (Section 7.A).

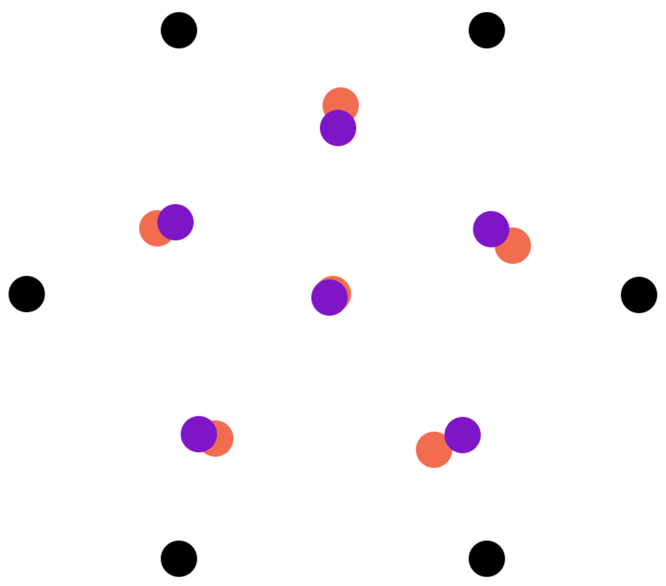


Figure 7.3: The result of optimizing the positions of 6 refined particles using our discretized problem (purple) and the continuous form (red), which results in a small difference, demonstrating a good approximation through our proposed discretization.

7.6.1 Spatial derivatives

Even though particle refinements can interfere with each other, leading to a global optimization problem, we refine particles separately with respect to their current neighborhood, see [OK12]. Thus only derivatives based on refined particles for each original particle need to be considered, and $\mathcal{N}$ can be assumed to be constant. This also allows for the optimization of all individual refinement steps in parallel. We thus need to consider the derivative of $\mathbb{E}$ with respect to the position of every inserted particle. In general a kernel function can be written as [DA12]

$$W_{ij} = W\left(\|\mathbf{x}_{ij}\|, h_{ij}\right) = \frac{C}{h_{ij}^d}\hat{W}\left(q\right), \tag{7.10}$$

where $h_{ij} = \frac{h_i+h_j}{2}$, $q = \frac{\|\mathbf{x}_{ij}\|}{h_{ij}}$, C is a normalization constant, d is the spatial dimensionality of the simulation and $\hat{W}(q)$ being the kernel function, i.e.,$\hat{W}(q) = [1-q]_+^3 - 4[0.5-q]_+^3$ for the cubic spline kernel, where $[\cdot] = \max(\cdot, 0)$. The derivative of the kernel function W with respect to the position of a particle i can then be calculated as

$$\nabla_i W_{ij} = \frac{\mathbf{x_{ij}}}{|\mathbf{x_{ij}}|}\frac{C}{h^{d+1}}\frac{\partial\hat{W}(q)}{\partial q} = \hat{\mathbf{x}}_{ij}W'_{ij}, \tag{7.11}$$

with $W'(r,h) = \frac{C}{h^{d+1}}\frac{\partial\hat{W}(q)}{\partial q}$ and $\hat{\mathbf{x}}_{ij} = \frac{\mathbf{x_{ij}}}{|\mathbf{x_{ij}}|}$. Using some linear algebra, shown in detail in the appendix 7.A, the spatial derivatives of the discretized error term for neighboring particle positions $\mathbb{E}_{\mathcal{N}}$ and refined particle positions $\mathbb{E}_{\mathcal{S}}$ with respect to the position of a refined particle i is given as

$$\nabla_i\mathbb{E}_{\mathcal{N}} = \sum_{n\in\bar{\mathcal{N}}} m_n\left(\tau_i + \tau_n\right)\nabla_i W_{in}, \tag{7.12}$$

with $\bar{\mathcal{N}} = \mathcal{N} \setminus o$, and

$$\nabla_i \mathbb{E}_{\mathcal{S}} = \sum_{s\in\mathcal{S}} m_s \left(\tau_i + \tau_s\right) \nabla_i W_{is}. \tag{7.13}$$

7.6.2 Mass distribution derivatives

For the efficient formulation of our optimization cost function, we utilize a set of tunable weights

$$\Lambda = \left[\lambda_0, \dots \lambda_{n-1}\right], \tag{7.14}$$

where $\frac{1}{\lambda_i}$ describes the individual mass ratio for each inserted particle i, with the constraint $\sum_s \frac{1}{\lambda_s} = 1$, i.e.,$m_s = m_o/\lambda_i$. Plugging these weights into the error terms τ_n (Eqn. 7.5) and τ_s (Eqn. 7.6) yields

$$\begin{aligned} \tau_n &= m_o \sum_{s\in\mathcal{S}} \frac{1}{\lambda_s} W_{ns} - m_o W_{no}, \\ \tau_s &= \sum_{j\in\mathcal{N}} m_j W_{sj} + m_o \sum_{k\in\mathcal{S}} \frac{1}{\lambda_k} W_{sk} - \rho_o. \end{aligned} \tag{7.15}$$

To calculate the derivative of the overall error terms with respect to these weights, we first need to consider the derivative of the kernel function with respect to the support radius, which can be written as

$$\frac{\partial W_{ij}}{\partial h_i} = -\frac{1}{2}\left[\frac{d}{h_{ij}} W_{ij} + q W'_{ij}\right], \tag{7.16}$$

which is a term not commonly found in computer animation; see appendix 7.A for further details. This can be used to evaluate the derivative of the kernel function with respect to the mass of a particle, which yields

$$\frac{\partial W(\mathbf{x}_{ij}, h)}{\partial m_i} = \frac{\partial W(\mathbf{x}_{ij}, h)}{\partial \mathbf{x}_{ij}} \frac{\partial \mathbf{x}_{ij}}{\partial m_i} + \frac{\partial W(\mathbf{x}_{ij}, h)}{\partial h} \frac{\partial h}{\partial m_i}, \tag{7.17}$$

where $\frac{\partial \mathbf{x}_{ij}}{\partial m_i} = 0$. The term $\frac{\partial W(\mathbf{x}_{ij},h)}{\partial h}$ can be determined using Eqn. 7.16, with $h = \frac{h_i + h_j}{2}$. Using the definition of the support radius $h_i = \eta \sqrt[3]{\frac{m_i}{\rho_i}}$ [Mon05] and applying the derivative, the missing term is given as

$$\frac{\partial h_i}{\partial m_i} = \frac{1}{3}\eta \left(\frac{m_i}{\rho_i}\right)^{-\frac{2}{3}} \frac{\partial \frac{m_i}{\rho_i}}{\partial m_i} = \frac{1}{3}\eta \left(\frac{m_i}{\rho_i}\right)^{-\frac{2}{3}} \frac{\frac{\partial m_i}{\partial m_i}\rho_i - m_i \frac{\partial \rho_i}{\partial m_i}}{\rho_i^2}, \tag{7.18}$$

where $\frac{\partial m_i}{\partial m_i}$ is 1. Applying $\frac{\partial}{\partial m_i}$ further to the standard SPH estimate for density $\rho_i = \sum_j m_j W_{ij}$ yields

$$\frac{\partial \rho_i}{\partial m_i} = \sum_j \left(\frac{\partial m_j}{\partial m_i} W_{ij} + m_j \frac{\partial W_{ij}}{\partial m_i}\right), \tag{7.19}$$

where $\frac{\partial m_j}{\partial m_i} = \delta_{ij}$ (Kronecker Delta). Putting these equations together results in

$$\frac{\partial W_{ij}}{\partial m_i} = \frac{\partial W_{ij}}{\partial h}\left[\frac{h_i}{3m_i} + \frac{h_i}{3\rho_i}\left(W_{ii} - \sum_k m_k \frac{\partial W_{ik}}{\partial m_i}\right)\right], \tag{7.20}$$

which means that the value for $\frac{\partial W_{ij}}{\partial m_i}$ depends on itself and values of neighboring particles. In theory this could be solved iteratively, similar to the relation of density and support radius, but we opt to follow the common computer animation notion of all particles having a support radius solely based on their rest density, i.e., $h_i = \eta\sqrt[3]{\frac{m_i}{\rho_0}}$ instead of $h_i = \eta\sqrt[3]{\frac{m_i}{\rho_i}}$ and, accordingly, $V_i = V_j \implies h_i = h_j$. This means that

(7.21) $$\frac{\partial W_{ij}}{\partial m_i} = \frac{h_i}{3m_i}\frac{\partial W_{ij}}{\partial h_i}.$$

Calculating the derivative of the kernel function with respect to a mass weight additionally requires the derivative of a weight λ_j by another weight λ_i. If in the optimization the weight of one particle is increased and all other particles are equally decreased, we can find a derivative

(7.22) $$\frac{\partial}{\partial \lambda_i}\left[1 - \sum_{i \neq j}\frac{1}{\lambda_i}\right] = \begin{cases} \frac{1}{n-1}\frac{1}{\lambda_i^2}, & i \neq j \\ 0, & \text{else.} \end{cases}$$

This can be expressed more generally by introducing a matrix $\mathbb{M}$, describing the distribution of weights, which for Eqn. 7.22 yields

(7.23) $$\mathbb{M}_{ij} = \begin{cases} \frac{1}{n-1}, & i \neq j \\ -1, & \text{else.} \end{cases}$$

Note that each column of the matrix should sum to zero to ensure mass conservation during optimization. Additionally, for refined particles the support radius does change with respect to the change of mass of a refined particle. For two particles s and j, this results in the following term

(7.24) $$\frac{\partial h_{sj}}{\partial \lambda_i} = \frac{1}{2}\left(\frac{\partial h_s}{\partial \lambda_i}\mathbf{1}_{\mathcal{S}}(s) + \frac{\partial h_j}{\partial \lambda_i}\mathbf{1}_{\mathcal{S}}(j)\right),$$

with $\mathbf{1}$ being the indicator function defined as

(7.25) $$\mathbf{1}_{\mathcal{S}}(j) = \begin{cases} 1, & j \in \mathcal{S} \\ 0, & \text{else.} \end{cases}$$

Finally, we can find the derivatives of the discrete error terms for neighboring positions and refined positions, respectively, are calculated as (see Appendix 7.A).

(7.26) $$\begin{aligned} \frac{\partial \mathbb{E}_{\mathcal{N}}}{\partial \lambda_i} &= \sum_{n \in \bar{\mathcal{N}}} m_n \left[\tau_n \frac{\mathbb{M}_{is}}{\lambda_i^2} W_{ns} + \frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial \lambda_i}(\tau_n + \tau_s)\right], \\ \frac{\partial \mathbb{E}_{\mathcal{S}}}{\partial \lambda_i} &= \sum_{k \in \mathcal{S}} m_s \left[\tau_s \frac{\mathbb{M}_{ik}}{\lambda_i^2} W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial \lambda_i}\tau_s\right], \end{aligned}$$

where

(7.27) $$\begin{aligned} \frac{\partial W_{sj}}{\partial \lambda_i} &= \frac{1}{6\lambda_i^2}\left(h_s\lambda_s\mathbb{M}_{is}\mathbf{1}_{\mathcal{S}}(s) + h_j\lambda_j\mathbb{M}_{ij}\mathbf{1}_{\mathcal{S}}(j)\right)\frac{\partial W_{sj}}{\partial h_{sj}}, \\ \frac{\partial W_{sj}}{\partial h_{sj}} &= -\frac{1}{h_{sj}}\left(dW_{sj} + |\mathbf{x}_{sj}|W'_{sj}\right), W'(r,h) = \frac{C}{h^d}\frac{\partial \hat{W}(q)}{\partial q}. \end{aligned}$$

7.7 Optimization methods

The main goal of the introduction of the discretized objective function and its partial derivatives in Section 7.6 is to determine optimal particle positions and masses for the refinement of fluids, irrespective of specific kernel functions or neighborhood requirements. The naïve solution is to start with a random distribution of positions with an equal amount of mass per inserted particle. However, optimizing from this starting point is too expensive to be done online during a simulation. Instead, we propose optimizing the refinement patterns a priori for ideal conditions, e.g.,in an isotropic hexagonal particle distribution, and use the resulting patterns as initialization for an online optimization.

7.7.1 A priori optimization

In order to generate a generic set of refinement patterns, we assume that an arbitrary particle o has an isotropic hexagonal neighborhood of particles $\mathcal{N}$. Therefore, we can simply apply our optimization to this ideal neighborhood and the inserted refined particles. We utilize an original particle with the properties

$$h_o = 1, \quad V_o = \frac{4}{3}\pi\frac{1}{N_h}, \quad r_o = \sqrt[3]{\frac{1}{N_h}}, \tag{7.28}$$

as this allows us to easily rescale the generated patterns for a particle with radius r, by scaling the generated pattern by r. We also separate the optimization for positions and masses into two distinct processes for efficiency. Using the analytical partial derivatives, from Sec 7.6.1 and 7.6.2, it is fairly straightforward to apply any optimization algorithm, e.g.,L-BFGS-B from SciPy [JOP+01], to optimize the positions of splitting patterns by stacking the components of the inserted particles' position $[\mathbf{x}_{0,x}, \mathbf{x}_{0,y}, ..., \mathbf{x}_{n-1,z}]$, which for n particles results in $n \cdot d$ variables. Figure 7.4 shows example patterns generated for 4, 8, 16 and 32 particles, respectively, in a 2D setting and Figure 7.5 shows patterns for 3D settings, which were all initialized with random particle distributions. Figure 7.4 shows that the error for the resulting configuration is mainly reduced at the particle positions, while it is rather high in between particles, where it has no practical influence.

Different kernel functions yield very similar spatial configurations, however, not all kernel functions converge equally fast, due to pairing instabilities; see Dehnen and Aly [DA12] for a general discussion on the differences between kernel functions. Particles may move further from the center, even beyond the hexagonal packing distance, or move together, i.e.,they pair, during the optimization. However, in either case, which can be identified easily, we restart the optimization with a different random initialization to avoid these local minima.

For the optimization of the mass distribution we utilize the SLSQP optimization method from SciPy [JOP+01]. However, other minimizers can be used as well, as long as the optimizer can handle the required constraint, i.e.,non-negativity of masses and mass conservation.

7.7.2 Online optimization

Applying the optimization methods described for the a priori optimization to online optimizations would be too expensive due to computational costs, and as such we

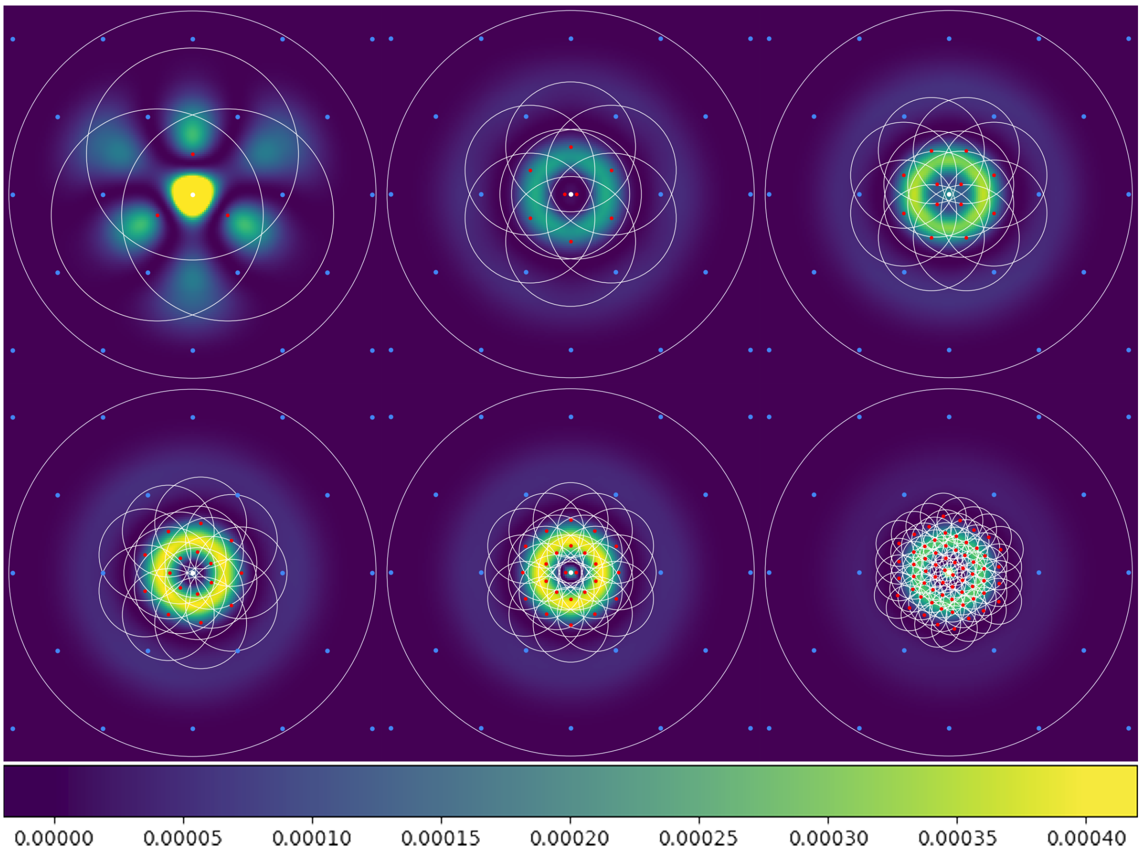


Figure 7.4: Optimized patterns for 3 (top-left), 8 (top-middle), 12 (top-right), 16 (bottom-left), 24 (bottom-middle) and 64 (bottom-right) particles, in 2D for the cubic spline kernel, showing particles from $\mathcal{S}$ (red), $\mathcal{N}$ (light blue) and the removed particle o (white). The coloring indicates τ, demonstrating that the error focuses mostly in regions that are not occupied by any particle.

aim to use simpler methods.

To optimize the positions we use a simple gradient descent algorithm. As we target GPUs for our optimizations, we use a number of threads (e.g., 96) per particle that should be refined, where we can parallelize the evaluation of $\mathbb{E}_{\mathcal{S}}$ and $\mathbb{E}_{\mathcal{N}}$. Here $\mathcal{N}$ denotes the actual set of neighbors of o. In addition we use a simple backtracking line search algorithm in order to determine the gradient step length γ, as we start from an already good initial guess. In our implementation we use up to 32 gradient steps with 8 backtracking attempts with an initial step length of $\gamma = 0.01$ and a backtracking weight $\beta = 0.5$.

The mass distribution of the particles, however, relies on a more complex optimization. The partial derivatives of the discretized objective function with respect to the mass ratios are significantly more complex which makes them expensive to evaluate. Furthermore, the memory used to calculate the Hessian, for some non linear constrained optimization methods, severely limits performance due to memory restrictions. As such, we chose to adapt an evolutionary optimization to our problem, which preserves mass conservation, while not requiring a gradient evaluation.

To avoid explicitly enforcing the constraint $\sum_{s\in\mathcal{S}} \frac{1}{\lambda_s} = 1$, we define a set of

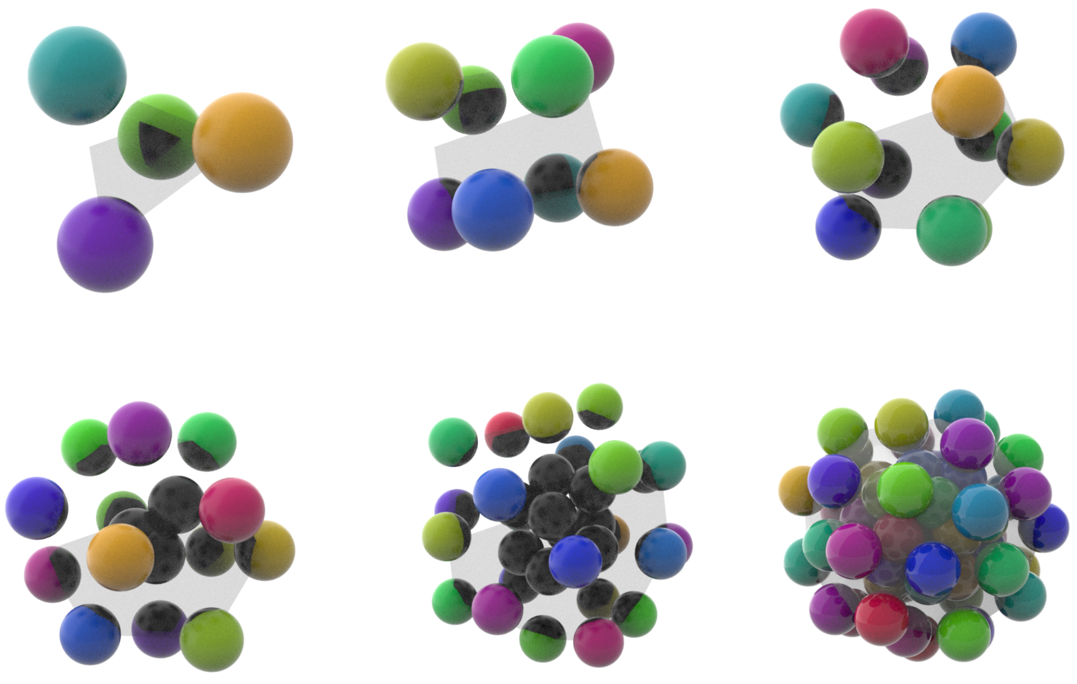


Figure 7.5: This figure shows the patterns for 4, 8, 12, 16, 32 and 64 particles in 3D for the Wendland 2 kernel (N=100) with the particle color being chosen for visual distinctiveness only. The transparent object visualizes the convex hull of the particle positions.

unnormalized values

$$\Phi = [\phi_0, \ldots, \phi_{n-1}], \tag{7.29}$$

from which the set of constrained weights is calculated as

$$\Lambda[i] = \frac{\phi_i}{\sum_{s\in\mathcal{S}} \phi_s}. \tag{7.30}$$

Note that Eqn. 7.30 enforces $\sum_s \frac{1}{\lambda_s} = 1$ by construction. To determine Φ we sample a normal distribution X, with mean $\bar{\mathbf{X}}$ and standard deviation σ, for every element as

$$\Phi[i] = \text{clamp}\left(X\left(\bar{X} = 1, \sigma^2 = 1\right), \frac{1}{2}, 2\right), \tag{7.31}$$

which we evaluate on every thread associated with a particle, e.g.,giving us 96 different sets. The values are clamped to avoid negative masses and very large differences between individual particle weights. We then evaluate the discretized objective function for all sets and determine the set Φ_b with the lowest error. Using this set we can then determine an updated set of weights

$$\Phi^{l+1}[i] = \text{clamp}\left(X\left(\bar{X} = \Phi_b[i], \sigma^2 = 2^{-l}\right), \frac{1}{2}, 2\right). \tag{7.32}$$

ALGORITHM 7.2: Our online optimization process applied to every particle o that is refined into n particles, executed in parallel on 96 threads.

```
Initialize positions of refined particles $\mathbb{X}$ using a priori pattern; Sec. 7.7.1
Initialize weights of refined particles $\Lambda$ to all be $n$; Eqn. 7.30
$e \leftarrow \mathbb{E}_\mathcal{N} + \mathbb{E}_\mathcal{S}$ using $\mathbb{X}$ and $\Lambda$; Eqn. 7.7
// Optimize positions using gradient descent
For $g \in [1, 32]$
    $\gamma \leftarrow 0.01$
    For $s \in [1, 8]$
        Evaluate $\nabla_i \mathbb{E}$ with $\mathbb{X}$ for $i \in \mathcal{S}$; Eqns. 7.12 and 7.13
        $\mathbb{X}^g[i] \leftarrow \mathbb{X}[i] + \gamma \nabla_i \mathbb{E}$
        $e^g \leftarrow \mathbb{E}_\mathcal{N} + \mathbb{E}_\mathcal{S}$
        If $e^g < e$
            Update $\mathbb{X} \leftarrow \mathbb{X}^g$ and stop $s$ iteration
        $\gamma \leftarrow \gamma \cdot \beta$
// Optimize weights using evolutionary optimization
Initialize $\Phi[i] = 1, \forall i \in \mathcal{S}$
$\Lambda_b[i] \leftarrow 1/n$
$e \leftarrow \mathbb{E}_\mathcal{N} + \mathbb{E}_\mathcal{S}$ using $\Lambda_b$ and $\mathbb{X}$
For $l \in [1, 8]$
    For every thread $t$
        Sample $\Phi_t^l$ using Eqn. 7.32
        $\Lambda_t^l[i] = \Phi_t^l[i] / \sum_{s \in \mathcal{S}} \Phi_t^l[s]$
        $e_t^l \leftarrow \mathbb{E}_\mathcal{N} + \mathbb{E}_\mathcal{S}$ using $\Lambda_t^l$ and $\mathbb{X}$
    Find thread with lowest error $t = \inf_t e_t^l$
    If $e_t^l < e_b$
        Update $\Phi \leftarrow \Phi_t^l, \Lambda_b \leftarrow \Lambda_t^l, e_b \leftarrow e_t^l$
```

We repeat this process for 8 iterations as a variance $\sigma^2 \approx 0.004$ has no practical influence on the result. We also always consider $\Phi = [1, \dots, 1]$, as this set of values describes a uniform distribution of mass. Note that this process is similar to a general random optimization and evolutionary optimization techniques but with modifications to enforce a constraint and to be parallelizable. The overall online optimization process is shown in Algorithm 7.2.

7.8 Error smoothing

Our minimization process, as will be shown later in the results in Section 7.10.1, cannot reduce the refinement error to zero, even for ideal particle distributions. Thus, we need to utilize further measures, which can help reduce the instabilities caused by the refinement error as they are not negligible for incompressible fluid simulations. In order to achieve this, we will first introduce a non-constant temporal blending method, followed by an extension of existing artificial viscosity methods that introduces local viscosity on newly refined particles.

7.8.1 Non-constant temporal blending

The basic idea of a temporal blending method [OK12] is that quantities are the result of a linear blending operation between the actual quantities A_s of refined particles $s \in \mathcal{S}_o$ and an estimated quantity for the original particle $\hat{A}_o$ as

$$A_s^{\text{blended}} = (1-\beta_s)A_s + \beta_s \hat{A}_o, \forall s \in \mathcal{S}_o, \tag{7.33}$$

where β describes a linear interpolation weight. In order to estimate the quantity for the original particle Winchenbach et al. [WHK17] track the position where the original particle would be at a new time point $t+\Delta t$ as $\boldsymbol{x}_o$ using the average velocity of all particles refined from o as

$$\boldsymbol{x}_o^{t+\Delta t} = \boldsymbol{x}_o^t + \Delta t \frac{1}{n} \sum_{s \in \mathcal{S}_o} v_s^t, \tag{7.34}$$

where n is the number of refined particles from o. Using this estimated position and the standard SPH estimate from Eqn. 7.1, we can determine an estimated quantity $\hat{A}_o$ by ignoring all particles refined from o and adding the interaction of o with itself

$$\hat{A}_o = m_o \frac{A_o}{\rho_o} W_{oo} + \sum_j \begin{cases} m_j \frac{A_j}{\rho_j} W_{oj}, & j \in \mathcal{N} \setminus \mathcal{S}_o, \\ 0, & j \in \mathcal{S}_o \end{cases}. \tag{7.35}$$

Winchenbach et al. [WHK17] additionally apply a clamping operation to estimated density values $\hat{\rho}_o$. The blending weight as described by Orthmann and Kolb [OK12] and Winchenbach et al. [WHK17] is given as

$$\beta_s^{t+1} = \beta_s^t + \Delta\beta, \quad \beta^0 = 1, \tag{7.36}$$

where a constant change in blend weight per time step t is utilized. The initial blend weight is 1, with a fixed change of blend weight per time step of $\Delta\beta = -\frac{1}{\Theta}$. Here Θ is usually chosen to be 10 and denotes the number of time steps over which blending occurs. Instead we propose to utilize the following blend weight β for any particle i as

$$\beta_i = \text{clamp}\left(\frac{\Delta t^0}{2\Delta t}\left[1 - \frac{t_i}{\Theta \Delta t^0}\right], 0, \frac{1}{2}\right) \tag{7.37}$$

which bases the blend weight β_i on a value describing the lifetime of a particle t_i, the current time step Δt, the time step at the time of refinement Δt^0, as well as the number of blend steps Θ. For a fixed time step this results in a linearly decreasing weight, per time step, as prior methods, but instead of starting with an initial blend weight of 1 we start with a blend weight of $\frac{1}{2}$. However, if the time step changes during the blending process, i.e., $\frac{\partial \Delta t}{\partial t} \neq 0$, the blend weight gets adjusted as well. For increasing time steps, $\frac{\partial \Delta t}{\partial t} > 0$ the blend weight decreases more slowly. For example with $\Delta t^0 = 0.1$ and $\Theta = 10$, changing the time step to $\Delta t = 0.2$ halfway through the blend process causes the blend weight to change from $\beta = \frac{1}{4}$ to $\beta = \frac{1}{8}$. For decreasing time steps $\frac{\partial \Delta t}{\partial t} < 0$ the blend weight instead reverts to $\beta = \frac{1}{2}$. This is motivated by the fact that $\frac{\partial \Delta t}{\partial t} > 0$ indicates a stabilizing overall simulation, whereas $\frac{\partial \Delta t}{\partial t} < 0$ indicates a destabilizing simulation.

7.8.2 Local viscosity

In general, the refinement error alters the local density in an incompressible fluid, causing visually noticeable instabilities. This divergence can be smoothed by introducing a higher artificial viscosity. However, increasing the artificial viscosity on a global level for all particles prevents the simulation of an overall relatively inviscid liquid. To avoid this problem, we only introduce additional artificial viscosity locally, which only affects particles that are in the process of being blended, e.g., those with $\beta_i > 0$. XSPH [Mon02] uses artificial viscosity to modify the velocity of a particle i, which is given as

$$v_i^{\text{new}} = v_i + \sum_j c \frac{m_j}{\rho_j} v_{ij} W_{ij}, \tag{7.38}$$

where c is the viscosity factor. We propose a modified c^{new} given by

$$c^{\text{new}} = c \begin{cases} 1, & i \in \mathcal{S}_o \wedge j \in \mathcal{S}_o \\ \left(1 + 0.5\frac{\beta_i + \beta_j}{2}\right), & \text{else} \end{cases}, \tag{7.39}$$

which can similarly be applied to a traditional artificial viscosity formulation [Mon05] by changing the corresponding viscosity factor ν. This term still results in the same global viscosity applied to all particle interactions c, but introduces an additional local viscosity $c\frac{(\beta_i+\beta_j)}{4}$ that only affects interactions of particles with newly refined particles, as β is 0 for any non-blending particle, and excludes interactions between refined particles belonging to the same original particle. This additional term has a maximal magnitude of $0.5c$, i.e.,it increases viscosity locally by at most 50%. Applying this artificial viscosity in our experiments reduces any instabilities that remain after our optimized refinement process, without noticeably changing the global behavior.

7.9 Sizing-functions

In adaptive SPH a sizing function determines the ideal particle volume $V(\boldsymbol{x})$ at a location $\boldsymbol{x}$ and is commonly defined using a distance function $d(\boldsymbol{x})$ from the region of interest, e.g.,the fluid surface. Usually, the sizing function specifies a smooth gradient from a base volume V_{base} at a maximum distance d_{max} to the finest volume $V_{\text{fine}} = \frac{1}{\alpha} V_{\text{base}}$ at the fluid surface with a desired adaptive volume ratio α. Winchenbach et al. [WHK17] proposed a linear sizing function that scales the volume directly with the distance, i.e.,$V \propto d$, as

$$V(\boldsymbol{x}) = \left[\frac{1}{\alpha} + \frac{d(\boldsymbol{x})}{d_{\text{max}}}\left(1 - \frac{1}{\alpha}\right)\right] V_{\text{base}}. \tag{7.40}$$

Unless otherwise noted we use this formulation for all of our results as the linear sizing of volume yields high surface-resolutions at moderate overall particle counts. However, other sizing functions might also be used. A practical problem that arises for Eqn. 7.40 is that very high adaptive volume ratios, e.g.,$1 : 4,000$, can result in a very thin sheets of high-resolution particles at the surface, which does not provide the expected improvement in quality. This effect, however, can be avoided by either using deep fluid volumes to increase the thickness of the highest-resolution sheet, additional factors in the sizing, e.g.,camera visibility or closeness to an object of interest (see the **torus** scene), or by using a different sizing function. A straight forward replacement for the sizing function in Eqn. 7.40 is to scale the particle radius linearly, resulting in a cubic scaling of particle volume based on surface distance. This sizing function can be defined as

$$V(\boldsymbol{x}) = \frac{4}{3}\pi \left[\left(\frac{1}{\sqrt[3]{\alpha}} + \frac{d(\boldsymbol{x})}{d_{\text{max}}}\left(1 - \frac{1}{\sqrt[3]{\alpha}}\right)\right) r_{\text{base}}\right]^3. \tag{7.41}$$

However, this sizing function generates significantly more particles for the same adaptive volume ratio, when compared to Eqn. 7.40, resulting in a reduction of achieved adaptive volume ratios by a factor 100. For example, for a scene with an achievable adaptive volume ratio of $1 : 1,000,000$ using Eqn. 7.40, using Eqn. 7.41 would result in an achievable adaptive volume ratio of $1 : 10,000$, due to computational resource limitations.

7.10 Results and discussion

All simulations were run on a single Nvidia GeForce RTX 2080ti GPU with 11 GiB of VRAM, a 32 core AMD Ryzen 3970x with 64 GiB of RAM. Pressure solving was done using DFSPH [BK15] with XSPH [Mon05] for artificial viscosity, with fluid air phase interactions based on Gissler et al. [Gis+17], surface tension effects model from Akinci et al. [AAT13], with the vorticity refinement method of Bender et al. [Ben+17], dynamically adjusted time steps based on the CFL condition [Ihm+13] and the data handling model from Winchenbach and Kolb [WK19]. In all examples we set DFSPH to a density error of 0.01% and a divergence error of 0.1%. Renderings were done using a custom ray tracing program, with surface extraction based on the work of Yu and Turk [YT13]. Surface distance calculations were based on a modified approach of Horvath and Solenthaler [HS13]. We use the overall adaptive method of Winchenbach et al. [WHK17] in our evaluations, although our approach is not restricted to this adaptivity approach. We implemented our approach in the open source SPH framework openMaelstrom [Win19] using the boundary handling approach of Winchenbach et al. [WAK20]. For the a priori optimizations we use SciPy [JOP+01]

7.10.1 Test scenes

We evaluated our approach in seven scenes. The **inlet** scene involves a fluid inlet, surrounded by a box, emitting fluid into a basin, which is agitated by a moving wall on the opposing side of the inlet stream; see Fig. 7.1. The **corner dam break** scene involves an initial fluid volume located in one corner of the simulation domain colliding with a regular obstacle in the opposing corner of the domain; see Fig. 7.6. The **double dam break** scene involves two fluid volumes, initially located in opposing corners of the simulation domain, colliding with a simple cubical rigid object placed in the center; see Fig. 7.11. The **simple dam break** scene involves a fluid volume in a simple box shaped domain with no additional obstacles; see Fig. 7.12. The **stream** scene involves a fluid inlet in a simulation with no gravity;

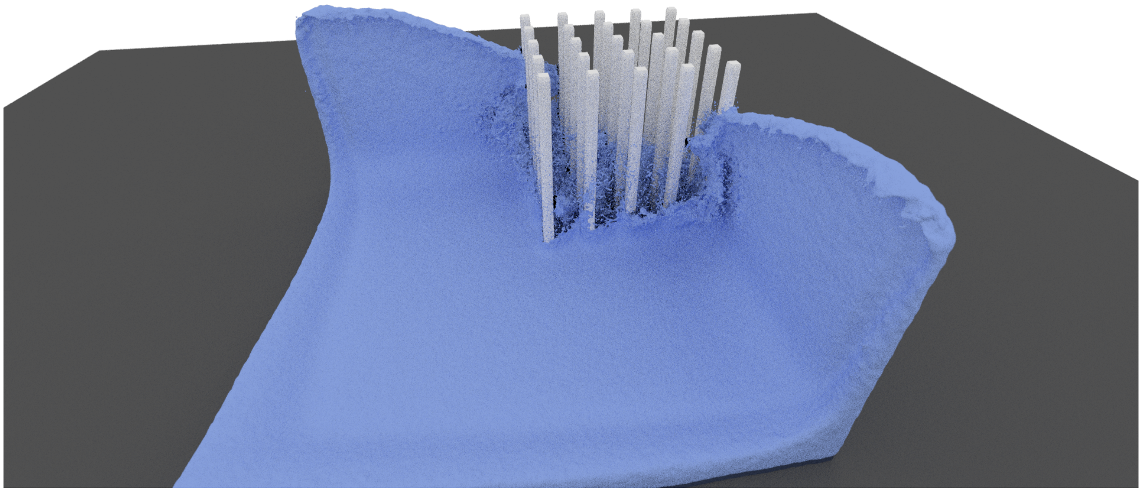


Figure 7.6: The **corner dam break** scene with an extracted fluid surfaced based on [YT13] for an adaptive volume ratio of $1 : 100$.

see Fig. 7.14. The **hemisphere dam break** scene involves a fluid volume colliding with a hemisphere on the floor; see Fig. 7.15. The **moving sphere** scene involves a solid sphere slowly being moved through a resting fluid volume; see Fig. 7.17. Finally, the **torus** scene involves a torus rotating about its vertical axis in a resting basin of liquid; see Fig. 7.16. We chose a basic particle radius of $r = 0.5m$ in all of our scenes, however our method would also work at different basic particle resolutions.

7.10.2 A priori position optimization

To evaluate our a priori optimization process, we optimized refinement patterns for 2 to 32 particles using the cubic-spline, quintic-spline, Wendland 2 and Wendland 4 kernels, using an L-BFGS-B optimizer [Byr+95]; see Fig. 7.7. The total error shows similar behavior for different kernel functions, i.e.,the refinement pattern for two particles has high error, with patterns around five to ten particles having lower errors, and increasing error ratios on higher particle counts. In our evaluation the cubic-spline function shows the largest error, an order of magnitude higher than the quintic-spline function, and the Wendland 4 Kernel to have similar behavior to the quintic-spline function. Additionally, the error on the neighboring particles is the main component of the overall error, i.e.,the error on the refined particles is two orders of magnitude smaller. Interestingly, the 1:13 splits results in icosahedron-shaped refinement patterns for all kernel functions, with one particle at the center, which is the configuration manually defined by Vacondio et al. [Vac+16].

7.10.3 A priori mass optimization

To evaluate the mass optimization process we use the patterns optimized a priori for positions only, see Sec. 7.10.2, and used a constrained trust-region optimizer

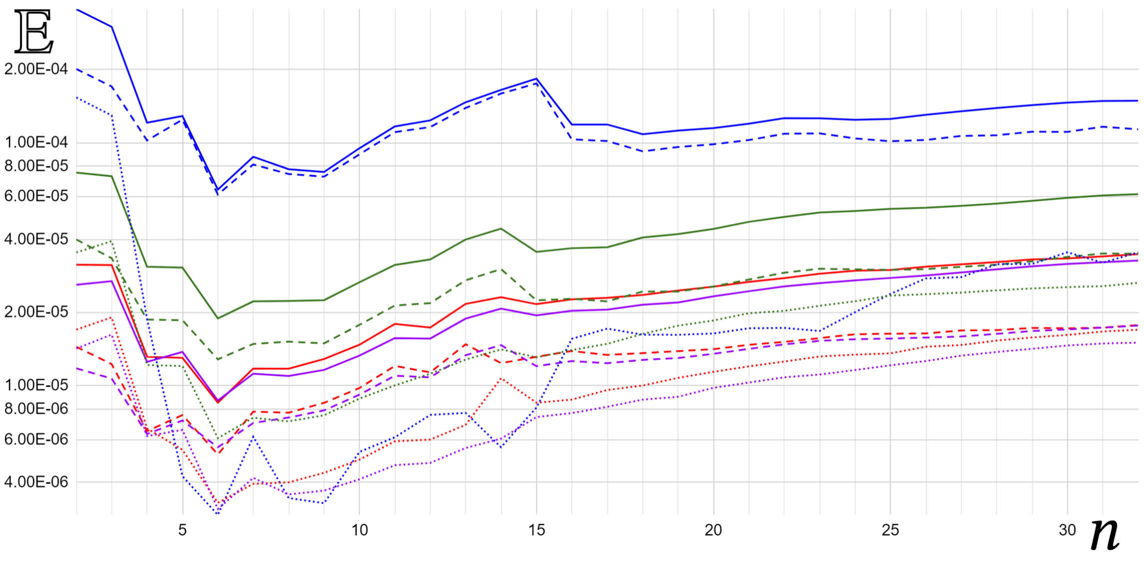


Figure 7.7: The logarithmic error terms $\mathbb{E}$ (solid), $\mathbb{E}_{\mathcal{N}}$ (dashed) and $\mathbb{E}_{\mathcal{S}}$ (dotted) for a $1 : n$ split using a cubic-spline (blue), quintic-spline (red), Wendland 2 (green) and Wendland 4 (purple) kernel functions in 3D after optimizing positions only.

[BSS87] to optimize the masses, without changing positions; see Fig. 7.8. The results are almost identical to the position optimization alone and do not show significant overall improvement for the kernels evaluated here.

7.10.4 A priori simultaneous optimization

After the a priori optimization of both positions and masses, we further optimized the refinement patterns, using a constrained trust-region and SLSQP optimizer [Kra88], by simultaneously optimizing positions and masses; see Fig. 7.9. The results show an overall reduction of the error by up to a factor of 2, mostly reducing the error on the neighboring particles and not on the refined particles themselves. The overall refinement patterns stay in very similar overall spatial configurations and get only slightly modified during the combined optimization.

We tried two different initialization schemes for the combined optimization, i.e., starting from pre-optimized spatial layouts and random initialization. While the pre-optimized initialization results in stable, but potentially local minima, the combined optimization did not robustly converge when initialized with random positions and masses, regardless of the optimization algorithm used. Furthermore, comparing our results against the prior refinement patterns of Winchenbach et al. [WHK17] (see Fig. 7.10), we can observe a significant reduction in the error terms. Overall, our refinement patterns provide an improvement of about two orders of magnitude, regarding both the error on refined and neighboring particles, and yield comparable errors across all refinement ratios.

7.10.5 Online optimization

Starting with the a priori optimized refinement patterns we used the **double dam break** scene to evaluate the errors during an SPH simulation, using the cubic-spline kernel and refinement patterns for 2 to 16 particles. In general, using the a

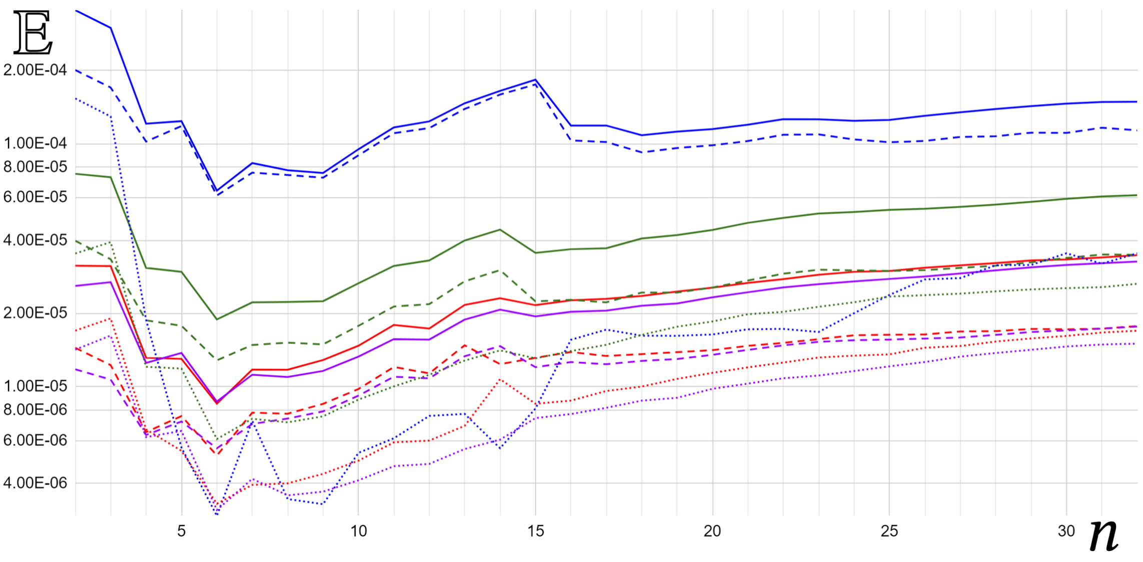


Figure 7.8: The logarithmic error terms $\mathbb{E}$ (solid), $\mathbb{E}_{\mathcal{N}}$ (dashed) and $\mathbb{E}_{\mathcal{S}}$ (dotted) for a $1 : n$ split using a cubic-spline (blue), quintic-spline (red), Wendland 2 (green) and Wendland 4 (purple) kernel functions in 3D after optimizing masses (with pre-optimized positions).

priori refinement patterns worked reasonably well in most cases. We, however, frequently observed instabilities, particularly in the fluid interior that resulted in local compression, which can destabilize the entire simulation. Applying our proposed online optimization avoids virtually all of these interior instabilities, however, instabilities caused by boundary interactions and by fully constrained particle merging remain (see also Sec. 7.4). Overall, applying the online optimization to practical particle configurations during a simulation results in slightly higher error values (in average an additional error of $\mathbb{E} \approx 0.01$) compared to the a priori optimization under ideal particle configurations. This additional error is not reduced when using the same optimization methods as a priori. Furthermore, outside of removing instabilities, the online optimization provides significant visual benefits on the fluid surface as using the exact same pattern on the surface leads to visibly repeating particle patterns on the surface, which are also visible in some surface extraction approaches. In the gravity-free **stream** scene, our method is capable of producing a smooth fluid surface in a difficult scenario and provides a smooth resolution gradient from low to high resolution, whereas prior methods were not able to stably simulate this scene, when enforcing incompressibility; see Fig. 7.14. Additionally, this improved behavior on the surface of an inlet flow allows us to emit particles at a fixed low resolution, as done in the **inlet** scene, and only refining the particles once they become visible. Moreover, evaluating the overall energy of the **simple dam break** scene, see Fig. 7.13, shows a significantly reduced loss of energy when using all aspects of our method, compared to prior work. Whilst using the online optimization alone, without local viscosity, does not provide significant benefits in this regard, i.e., the blue and green lines are fairly close, using our online optimization allows for the utilization of our local viscosity scheme that provides a significant reduction in dampening. Using the blending process, and refinement patterns from Winchenbach et al. [WHK17] results in a mostly stable simulation, but causes some instabilities. For example, at 12 sec-

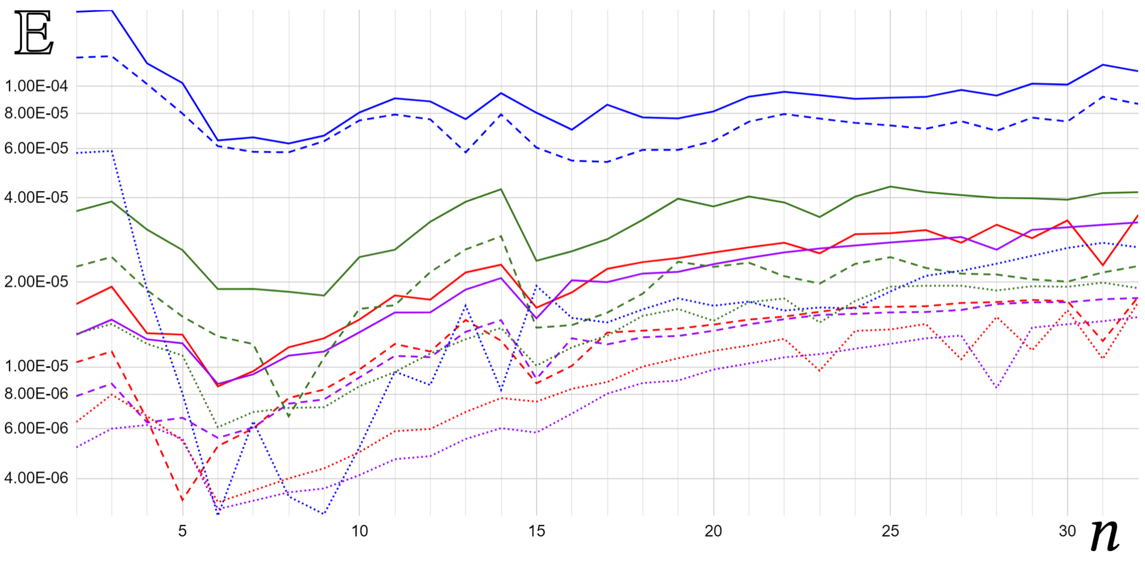


Figure 7.9: The logarithmic error terms $\mathbb{E}$ (solid), $\mathbb{E}_{\mathcal{N}}$ (dashed) and $\mathbb{E}_{\mathcal{S}}$ (dotted) for a $1 : n$ split using a cubic-spline (blue), quintic-spline (red), Wendland 2 (green) and Wendland 4 (purple) kernel function in 3D after simultaneous optimization (with pre-optimized positions and masses).

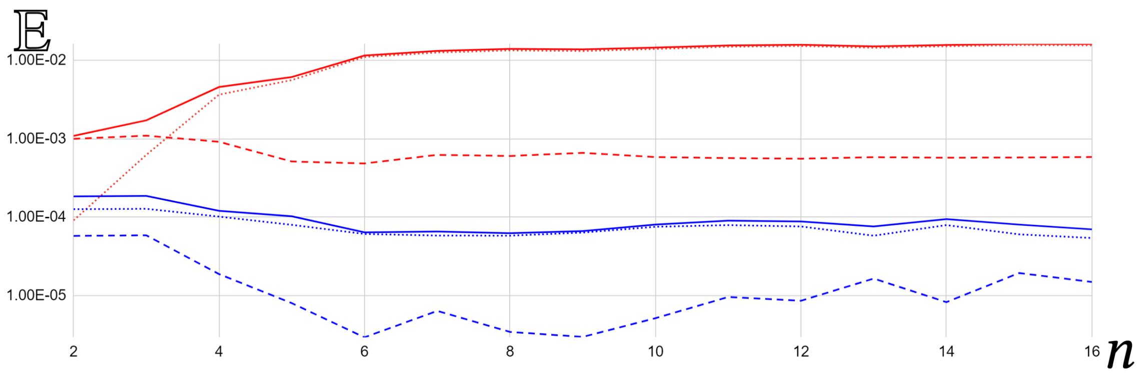


Figure 7.10: The logarithmic error terms $\mathbb{E}$ (solid), $\mathbb{E}_{\mathcal{N}}$ (dashed) and $\mathbb{E}_{\mathcal{S}}$ (dotted) for a $1 : n$ split for a cubic-spline kernel using our approach (blue) and manually tuned refinement patterns [WHK17].

onds (see Fig. 7.13) a momentary increase of the total energy can be observed, which was caused by an instability induced by a particle refinement. Reducing the impact of these momentary increases in energy can be achieved by increasing the artificial viscosity, however, this causes an overall significant loss of detail and a noticeably viscous fluid behavior; see Fig. 7.12 bottom right and top right. Note that increasing the artificial viscosity only reduces the impact of these instabilities but does not prevent them completely; see Fig. 7.15 top row. Furthermore, the **moving sphere** scene (see close-up Fig. 7.17) highlights the smooth transition of particles between resolutions, even as they are close to a boundary object, using our method.

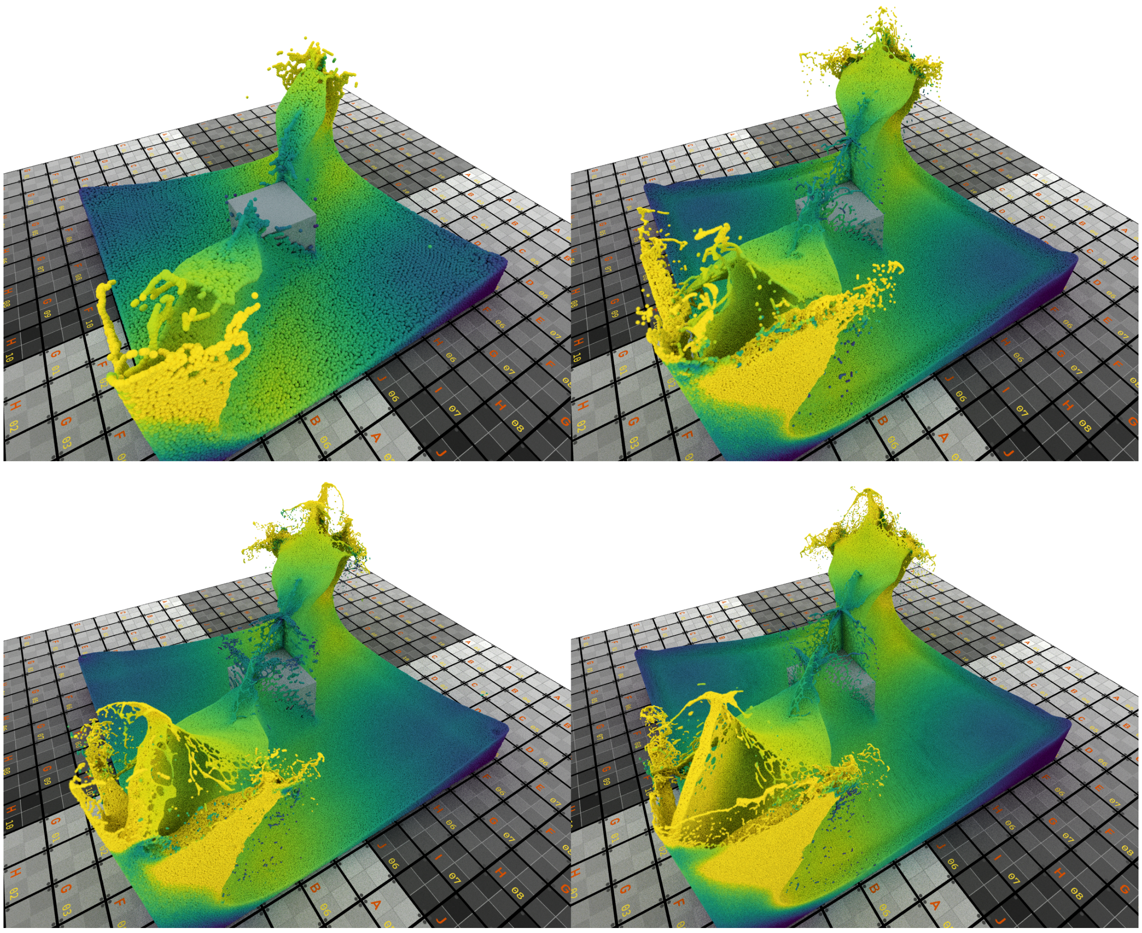


Figure 7.11: The **double dam break** scene we used to evaluate performance and online optimization, particle velocity color coded from purple (0 m/s) to yellow (30 m/s). Top left shows a low resolution simulation ($r = 0.5$m), bottom left an adaptive resolution simulation ($r_{\text{base}} = 0.5$m, 1:128 adaptive), top right a uniform simulation with the same particle count as the adaptive simulation ($r = 0.218$m) and bottom right a high-resolution simulation ($r = 0.1$m).

7.10.6 Influence of local viscosity and blending

In order to evaluate the influence of our blending scheme and the local viscosity approach we use the **simple dam break** scene. We visually compare the overall flow behavior of using our blending with local viscosity and online optimization, our blending and online optimization, our blending and the blending approach of Winchenbach et al. [WHK17] (using their refinement patterns). Comparing our blending against the prior approach, we only observe a small difference (Fig. 7.12 bottom left and bottom right). Adding the online optimization allows us to lower the overall viscosity of the simulation, as it becomes more stable, resulting in more flow details. However, adding the local viscosity allows us to reduce the global artificial viscosity by a factor of 2, resulting in a significant increase of more surface details. In the **hemisphere dam break** scene the induced instabilities from the refinement process using prior approaches are too high to be compensated by an increased artificial viscosity and, thus, fully destabilize the simulation at higher adaptive volume ratios; see Fig. 7.15 top right. In contrast to this, our online optimization process ensures a low refinement error and, accordingly, enables much

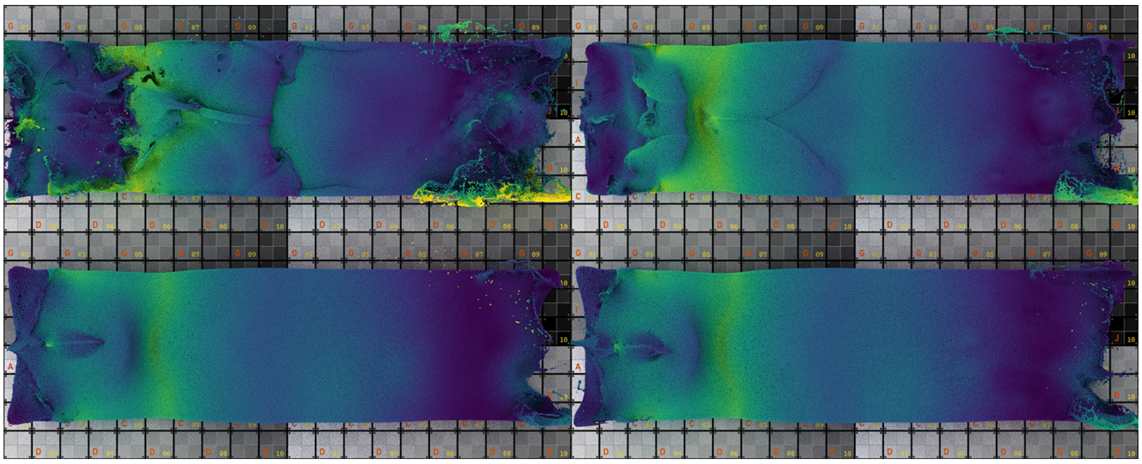


Figure 7.12: The **simple dam break** scene we used to evaluate the impact of our proposed blending, local viscosity and online optimization approaches, particle velocity color coded from purple (0 m/s) to yellow (30 m/s). Top left uses our blending, local viscosity and online optimization, top right uses our blending and online optimization, bottom left uses our blending and bottom right uses the approach of [WHK17]. Velocity color coded.

Variant	n_{fluid}	**radius** $/m$	**ratio**	Δt $/ms$	**Frame** $/s$	**Adaptive** $/ms$
Low	168K	0.5	1:1	8.0	0.15	
Average	1.9M	0.218	1:1	5.8	1.55	
High	21M	0.1	1:1	2.0	45.0	
Adaptive	1.9M	0.5	1:128	3.8	1.77	325

Table 7.1: Quantitative comparison for the **double dam break** scene; see Fig. 7.11. All performance numbers are average values with respect to 30 seconds simulation time and timings refer $1/60$s of simulation time.

higher adaptive volume ratios in difficult scenarios.

7.10.7 Performance

To evaluate the performance and efficiency of our method we first use the **double dam break** scene; see Fig. 7.11. In this scene, we compare the adaptive simulation with a volume ratio of 128:1, a fixed resolution simulation with the same number of particles as the adaptive simulation, on average, and a high-resolution simulation at approximately the finest resolution of the adaptive simulation. The simulated time is 30 sec; see Table 7.1 for the quantitative results. Overall, our method provides comparable performance to a simulation of equal particle count, with some overhead due to the usage of an adaptive method. Note that the time per frame of our method minus the time spent on adaptivity related methods is less than the time per frame of the average resolution simulation due to a lower time step requiring fewer pressure solver iterations per frame. Compared to the high-resolution variant, our method with moderate adaptive volume ratios pro-

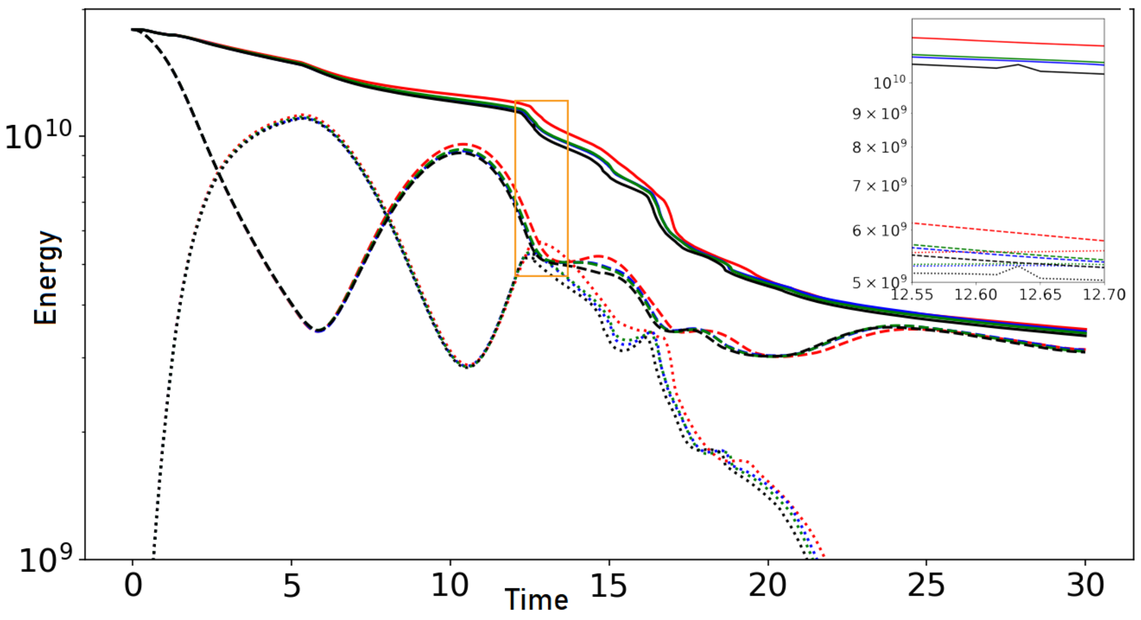


Figure 7.13: The overall energy for the **simple dam break** scene over 30 simulated seconds. The dashed line indicates potential energy, the dotted line kinetic energy and the solid line indicates total energy. The graph compares the prior approach from [WHK17] (black) against our approach with our blending, local viscosity and online optimization (red), with our blending and online optimization (blue) and with only our blending (green). The top right section shows the orange region closer up.

vides a significant speed-up of approximately 25 times. Accordingly, the speed-up will become significantly higher, at higher adaptive volume ratios. However, due to computational resource limitations, we were not able to provide a similar comparison against higher uniform resolution. Overall, the surface appearance of our method is similar to the high-resolution one, i.e.,considering the tearing of thin fluid sheets, but at orders of magnitude lower computational costs, even at moderate adaptive volume ratios. Furthermore, in the **torus** scene, our method can focus computational resources in small areas of interest, allowing for much greater detail without requiring hundreds of millions of particles in areas that are not of interest. However, scenes of very high adaptive volume ratios are difficult to render as the adaptivity induces a highly uneven particle distribution that causes raytracing acceleration structures, e.g., kd-trees, to be very unbalanced and, thus, inefficient. Accordingly, even when rendering particles as spheres, the computational requirements increase linearly with higher adaptive volume ratios, i.e., $\mathcal{O}(\alpha)$, and make rendering very high ratios computationally difficult. For example, with our computational resources, the **hemisphere dam break** scene required 4 hours to render a sequence of 30 seconds at an adaptive volume ratio of $1 : 1,000$, the **moving sphere** scene took 2.5 days to render at a ratio of $1 : 20,000$ for a 20 second sequence, whereas the **torus** scene required multiple hours for a single frame at a ratio of $1 : 1,000,000$.

Finally, at very high adaptive volume ratios, the neighborhood search becomes computationally increasingly expensive. For example, at a ratio of $1 : 1,000$, a cell contains approximately $12,000$ particles, compared to 12 particles per cell

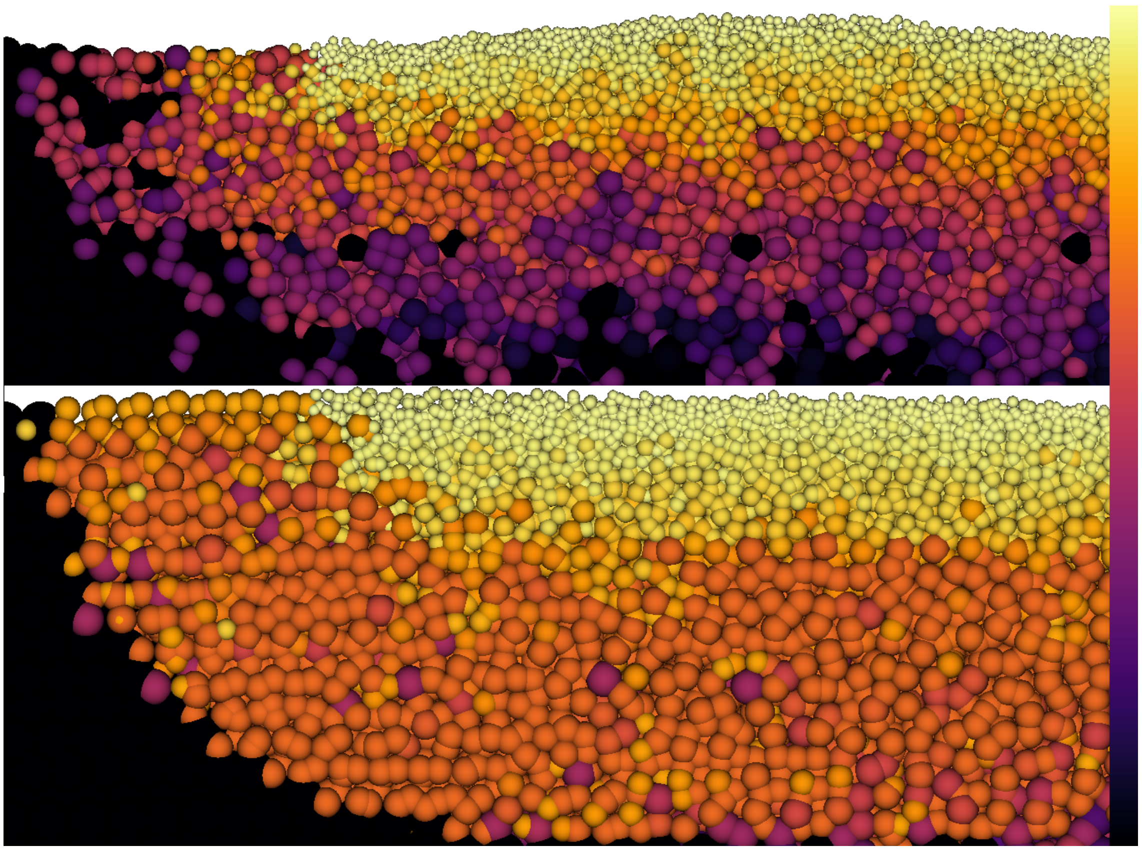


Figure 7.14: The gravity-free **stream** scene, particle volume color coded from black to yellow. The prior approach [WHK17] (top) is not able to stably simulate this scenario, whilst our improved method (bottom) only produces some slight irregular particle distributions.

in homogenous resolutions. The data-structure approach of Winchenbach and Kolb [WK19] resolves these problems in most situations; however, due to symmetric interactions of particles, required to ensure stability, the number of particles queried to find the actual neighbors of a particle can still be significantly higher than for homogenous resolutions. This problem can reduced by ensuring a large enough distance between low and high-resolution particles, e.g.,by setting d_{max} sufficiently high in Eqn. 7.40. Furthermore, limiting the number of neighbors per particle, see [WHK16] and [WK19], can further reduce the problem for actual SPH evaluations, however the neighborhood search is still a computationally expensive operation.

7.10.8 Clamping mass distributions

When using an adaptive method, the user specifies a desired volume ratio between the volume of the smallest V_{fine} to the volume of the largest V_{base} particles. However, this desired ratio is almost never exactly achieved. For example, a particle with $V = \frac{1}{350}V_{\text{base}}$ and a desired ratio of $1:1000$ might be split into 3 particles, which results in particles of volume $\frac{1}{1150}V_{\text{base}}$, exceeding the desired ratio. Optimizing the mass distribution exacerbates this effect as the optimization process creates particles of very different volumes. In Sec. 7.7.2, we proposed to clamp the weights

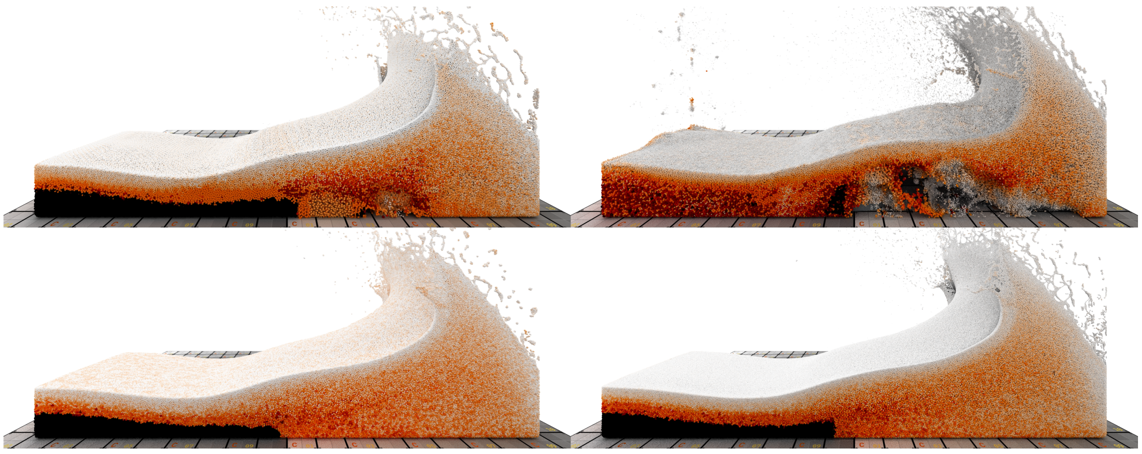


Figure 7.15: The **hemisphere dam break** scene, particle volume color coded from black to white. The top row uses the prior approach, while the bottom row uses our approach, with the left column using a $1:32$ adaptive volume ratio and the right column using a $1:1000$ adaptive volume ratio. Without an online optimization process (top row), instabilities appear at the boundary that get more pronounced as the adaptive volume ratio increases and would require significant added viscosity to reduce. Using our approach (bottom row), with its online optimization, yields a stable simulation for both adaptive volume ratios.

ϕ between 0.5 and 2, which limits the variation in particle sizes. In the simulation shown in Fig. 7.6 the clamped optimization process results in an effective ratio of $1:1250$ instead of the desired $1:512$ ratio. Not clamping the weights resulted in an effective ratio of $1:7500$. Consequently, this substantial difference in smallest particle volume requires a significantly smaller time step, due to the CFL condition.

7.10.9 Limitations

The adaptive method of Winchenbach et al. [WHK17], which we base our contributions on, already demonstrated scaling problems of certain parameters, e.g., the surface tension parameters used by Akinci et al. [AAT13] and parameters used for surface extraction by Yu and Turk [YT13]. That is, these parameters are heavily dependent on particle sizes, causing visual discontinuities in the surface extraction. Additionally, Winchenbach et al. [WHK17] described a problem with boundary handling methods based on particle representations, due to size differences, which can be avoided by using non-particle-based methods, i.e., [KB17]. Moreover, particle merging can lead to instabilities, especially close to boundaries, if applied with SPH solvers commonly used in computer animation. Furthermore, if the sizing function is based on the surface distance of a particle, i.e., using the surface-distance method of Horvath and Solenthaler [HS13], the stability of these methods plays an important role in the stability of the overall method. Accordingly, some artifacts may arise due to a non-stable sizing function, i.e., particles sitting on the surface of a boundary might not be properly detected as surface particles. Investigating this is beyond the scope of this paper.

Additionally, rendering a simulated fluid surface is an important aspect in computer animation. However, existing surface extraction methods are not designed for varying particle resolutions. They often involve parameters that significantly de-

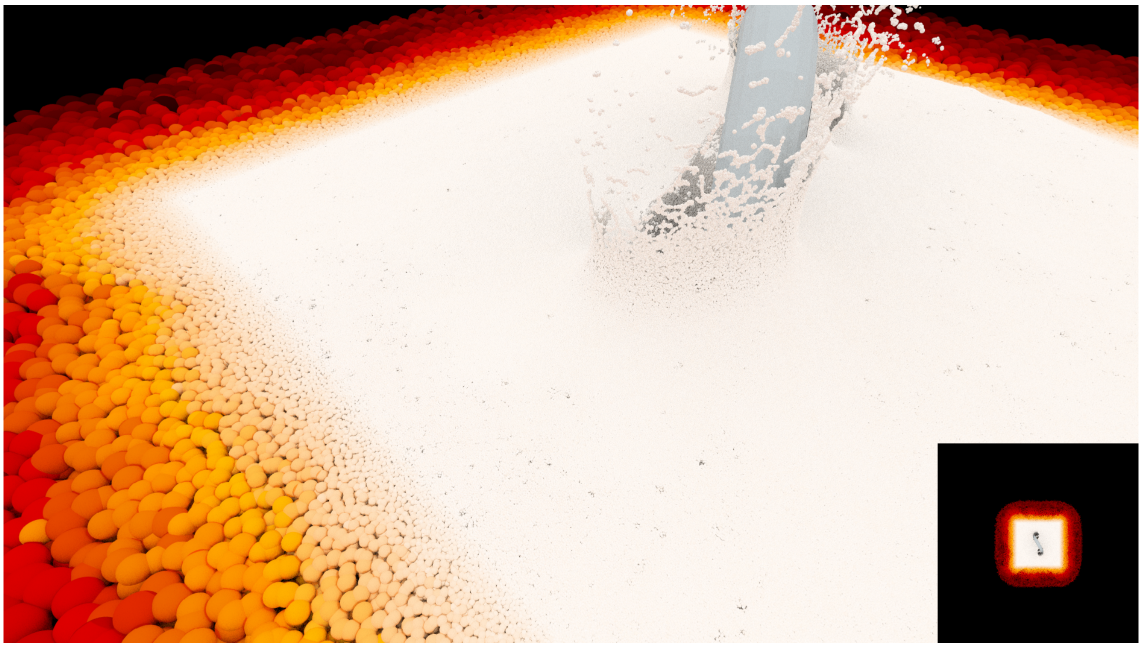


Figure 7.16: The **torus** scene, particle volume color coded black to white, with the bottom right showing the overall simulation domain. Our method can simulate an adaptive volume ratio of up to $1 : 1,000,000$, allowing for fine details close to the torus but limiting overall computational resources.

pend on the particle resolution and lead to visual artifacts such as missing details in high-resolution areas, lumpy surfaces in low resolution areas or visible changes in areas of varying resolution. Because of this, and since we explicitly need to investigate the varying particle resolution, we opt for displaying particle-based renderings and only provide an example of a surface extraction for parameters chosen for the high-resolution surface; see Fig. 7.6. For very high adaptive volume ratios even particle-based renderings become impractical; see Fig. 7.16.

Please note that in the images we use linearly color coded quantities. Thus, a change from the highest particle volume to a particle with half the volume, i.e., a 1:2 refinement, results in a visual discontinuity. Nonlinear color mapping could resolve this discontinuity to some degree, but makes the results more difficult to interpret.

7.11 Conclusions

We presented an optimization approach for particle refinement patterns, based on a novel discretized objective function that describes the error introduced by the particle refinement for symmetric SPH formulations. This allows us to significantly improve stability and removes the need for user intuition and parameter tuning, and is applicable to arbitrary refinement ratios using any kernel function. Our optimization approach works both a priori, to generate refinement patterns for ideal particle distributions, and online, to optimize the refinement pattern during a simulation with respect to the specific particle neighborhood. We also presented an improved non-linear blending process that, together with a novel local artificial viscosity formulation, that removes the impact of residual refinement errors. Our

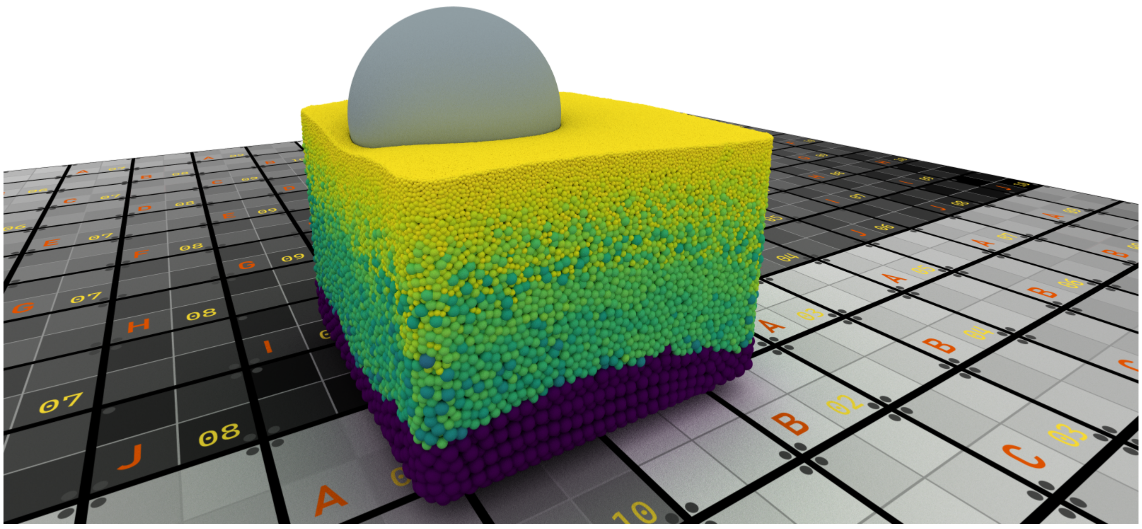


Figure 7.17: The **moving sphere** scene, particle volume color coded from high (purple) to low (yellow). Here a sphere slowly moves through a pool of liquid with an adaptive volume ratio of $1:20,000$.

improved process allows for the simulation of highly adaptive incompressible SPH flows, even in highly turbulent and low-viscosity situations. Currently, our approach is mostly limited by other methods it relies upon, i.e., surface extraction methods.

7.A Appendix

This document is intended to provided an additional, detailed, derivation of some of the aspects of the main paper content and was originally part of the supplementary material but is included as an appendix here for completeness.

7.A.1 Problem statement

In order to determine the error terms, and their derivatives, of our contribution we need various derivative terms of an SPH kernel function. As these terms are often hard to find, require special care to be taken when determining them or are a common source of notational differences, we will first discuss generic kernel functions and their derivatives before transitioning to the actual error terms. The kernel derivatives are no new contribution but are important to keep consistent.

7.A.2 Kernel functions

In SPH the most essential part is the kernel function. A kernel function W is used in the core formulation of SPH:

$$A(\mathbf{x}) = \sum_j A_j \frac{m_j}{\rho_j} W_{ij}. \tag{7.42}$$

This formulation allows one to determine the value of a quantity A at any point $\mathbf{x}$ using neighboring particles j and their respective quantity A_j, their corresponding mass m_j and their density ρ_j weighed by the kernel function W. A kernel function for SPH has to fulfill the following basic, mathematical, requirements:

- Converge to a Dirac delta function with 0 support: $\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}', \mathbf{h}) = \delta(\mathbf{x}-\mathbf{x}')$
- Integrate to one over the support domain:$\int_\Omega W(\mathbf{x}-\mathbf{x}', \mathbf{h})\mathbf{dx}' = \mathbf{1}$
- Have compact support: $W(\mathbf{x}-\mathbf{x}', \mathbf{h}) = \mathbf{0}; ||\mathbf{x}-\mathbf{x}'|| > \kappa\mathbf{h}$
- Be symmetric: $W(\mathbf{x}-\mathbf{x}', \mathbf{h}) = \mathbf{W}(\mathbf{x}'-\mathbf{x}, \mathbf{h})$
- Be even:$\nabla_{\mathbf{x}'} W(\mathbf{x}-\mathbf{x}', \mathbf{h}) = -\nabla_{\mathbf{x}}\mathbf{W}(\mathbf{x}-\mathbf{x}', \mathbf{h})$

The kernel should also be positive, monotonically decreasing and sufficiently smooth for good behavior in an SPH method. In practice many types of functions can achieve this goal, but the exact choice of function is not relevant here.

A generic kernel function can be written as:

$$W(\mathbf{x}-\mathbf{x}', \mathbf{h}) = \frac{\alpha_\mathbf{d}}{\mathbf{h}^\mathbf{d}}\hat{\mathbf{W}}(\mathbf{q}) \tag{7.43}$$

With d being the spatial dimension, α_d a scaling factor to satisfy $\int_\Omega W(\mathbf{x}-\mathbf{x}', \mathbf{h})\mathbf{dx}' = \mathbf{1}$, $q = \frac{||\mathbf{x}-\mathbf{x}'||}{h}$ a dimensionless number representing the relative distance and finally $\hat{W}$ a function that is defined over the range $[0,1]$. The scaling factor α_d is also notated as C if the dimensionality is clear from context.

$\hat{W}$ is sometimes referred to as a normalized kernel function, as it requires an input in the range $[0,1]$, however this might be misleading as sometimes $\bar{W}(\mathbf{x}-\mathbf{x}', \mathbf{h}) = \frac{\mathbf{W}(\mathbf{x}-\mathbf{x}',\mathbf{h})}{\mathbf{W}(\mathbf{0},\mathbf{h})}$ is also referred to as a normalized kernel function.

For an actual particle i we define the support radius h as

$$h_i = \eta\sqrt[3]{\frac{m_i}{\rho_i}}, \tag{7.44}$$

where η is a parameter that controls the number of neighbors per particle, m_i representing the mass of the particle and ρ_i being the actual density of the particle given as

$$\rho_i = \sum_j m_j W_{ij}. \tag{7.45}$$

At this point it is relevant to note a difference in notation between computer animation and computational fluid dynamics. The former uses h directly as the support radius, whereas the latter uses h as the smoothing length and H as the support radius, where H is determined by multiplying h by H/h, which for cubic spline kernels is ≈ 1.84. For our purposes we utilize the computer animation notation as this means a kernel function is defined based solely in the range $[0,h]$ instead of $[0,1.84h]$ for cubic spline kernels.

In order to ensure symmetric interactions, under adaptive resolutions, a symmetric support radius for an interaction of two particles i and j has to be chosen as

(7.46) $$W_{ij} = W(|\mathbf{x}_i - \mathbf{x}_j|, \frac{h_i + h_j}{2}),$$

where $\frac{h_i+h_j}{2}$ indicates this choice of a symmetric kernel function.

7.A.3 Spatial derivative $\nabla_i W_{ij}$

Applying the spatial derivative $\nabla_i = [\frac{\partial}{\partial x_i}, \frac{\partial}{\partial y_i}, \frac{\partial}{\partial z_i}]$ to the kernel function W results in

(7.47) $$\nabla_i W_{ij} = \frac{x_{ij}}{|x_{ij}|}\frac{C}{h^{d+1}}\frac{\partial \hat{W}(q)}{\partial q},$$

where $\frac{C}{h^{d+1}}\frac{\partial \hat{W}(q)}{\partial q}$ is sometimes also denoted as $\nabla W(r,h)$ with no subscript on ∇ in literature, however this can easily be misleading to readers so we avoid this notation and instead define

(7.48) $$W'(r,h) = \frac{C}{h^{d+1}}\frac{\partial \hat{W}(q)}{\partial q}.$$

Accordingly we can write $\nabla_i W_{ij}$ as

(7.49) $$\nabla_i W_{ij} = \hat{\mathbf{x}}_{ij} W'_{ij}.$$

Note, however, that this is just a notational short hand to make some terms more readable.

7.A.4 Derivative by support radius $\frac{\partial}{\partial h} W_{ij}$

The prior derivative by spatial quantities is relatively common in literature as it is required for many methods, i.e. pressure solvers. A derivative by support radius is much less commonly found as it is most often part of so called 'gradient-h' terms that are sometimes used in CFD, or astrophysical, applications of SPH that require these terms for an improved numerical accuracy. As this term is much less commonly used we will go into more detail on this derivative and how certain approximations have to be used for symmetric SPH.

The first step is to apply the derivative $\frac{\partial}{\partial h}$ to the kernel function, which yields

(7.50) $$\frac{\partial W(r,h)}{\partial h} = \frac{\partial \left[\frac{C}{h^d}\hat{W}(q)\right]}{\partial h},$$

where due to the product rule this expression can be expanded as

(7.51) $$\frac{\partial W(r,h)}{\partial h} = C\frac{\partial \frac{1}{h^d}}{\partial h}\hat{W}(q) + \frac{C}{h^d}\frac{\partial \hat{W}(q)}{\partial h}.$$

The first term can be simplified by re-substitution of the kernel function itself as

$$C\frac{\partial \frac{1}{h^d}}{\partial h}\hat{W}(q) = -\frac{d}{h}\frac{C}{h^d}\hat{W}(q) = -\frac{d}{h}W(r,h). \tag{7.52}$$

The second term can be simplified by applying $\frac{\partial}{\partial h}$ further as

$$\frac{C}{h^d}\frac{\partial \hat{W}(q)}{\partial h} = \frac{C}{h^d}\frac{\partial \hat{W}(q)}{\partial q}\frac{\partial q}{\partial h}, \tag{7.53}$$

with $q = \frac{r}{h}$. We can simplify $\frac{\partial q}{\partial h}$ by again applying the derivative to yield

$$\frac{\partial q}{\partial h} = r\frac{\partial \frac{1}{h}}{\partial h} = -r\frac{1}{h^2}. \tag{7.54}$$

Thus the second term, using a re-substitution of W', becomes

$$\frac{C}{h^d}\frac{\partial \hat{W}(q)}{\partial q}\frac{\partial q}{\partial h} = -q\frac{C}{h^{d+1}}\frac{\partial \hat{W}(q)}{\partial q} = -qW'(r,h), \tag{7.55}$$

which in total results in

$$\frac{\partial W(r,h)}{\partial h} = -\frac{d}{h}W(r,h) - qW'(r,h). \tag{7.56}$$

Applying the symmetric SPH formulation to W as $W\left(r, \frac{h_i+h_j}{2}\right)$, where effectively $h = \frac{h_i+h_j}{2}$, which, due to partial derivatives, gives the final derivative for the kernel function as

$$\frac{\partial W(r,h)}{\partial h_i} = \frac{1}{2}\frac{\partial W(r,h)}{\partial h} = \frac{1}{2}\left[-\frac{d}{h}W(r,h) - qW'(r,h)\right], \tag{7.57}$$

which for a specific interaction between two particles i and j can also be written as

$$\frac{\partial W_{ij}}{\partial h_i} = -\frac{1}{2}\left[\frac{d}{h_{ij}}W_{ij} + qW'_{ij}\right]. \tag{7.58}$$

7.A.5 Derivative by mass $\frac{\partial}{\partial m_i}W_{ij}$

In addition to the partial derivative of the kernel by the support radius, we also need the partial derivative by mass, $\frac{\partial}{\partial m_i}$. Applying this to the kernel function yields

$$\frac{\partial W(r,h)}{\partial m_i} = \frac{\partial W(r,h)}{\partial r}\frac{\partial r}{\partial m_i} + \frac{\partial W(r,h)}{\partial h}\frac{\partial h}{\partial m_i}. \tag{7.59}$$

We set $\frac{\partial r}{\partial m_i} = 0$ as the position is not directly influenced by the mass. This leaves $\frac{\partial W(r,h)}{\partial h}\frac{\partial h}{m_i}$ where $\frac{\partial W(r,h)}{\partial h}$ is treated as before. The remaining term then is $\frac{\partial h}{m_i}$. Recall that

$$h_i = \eta \sqrt[3]{\frac{m_i}{\rho_i}}, \tag{7.60}$$

where we apply the derivative by $\frac{\partial}{\partial m_i}$, which yields

$$\frac{\partial h_i}{\partial m_i} = \frac{1}{3}\eta \left(\frac{m_i}{\rho_i}\right)^{-\frac{2}{3}} \frac{\partial \frac{m_i}{\rho_i}}{\partial m_i} = \frac{1}{3}\eta \left(\frac{m_i}{\rho_i}\right)^{-\frac{2}{3}} \frac{\frac{\partial m_i}{\partial m_i}\rho_i - m_i \frac{\partial \rho_i}{\partial m_i}}{\rho_i^2}, \tag{7.61}$$

where $\frac{\partial m_i}{\partial m_i}$ is obviously 1. Refactoring results in

$$\frac{h_i}{3m_i} + \eta \frac{1}{3}\left(\frac{m_i}{\rho_i}\right)^{-\frac{2}{3}} \frac{m_i}{\rho_i^2}\frac{\partial \rho_i}{\partial m_i} = \frac{h_i}{3m_i} - \frac{h_i}{3\rho_i}\frac{\partial \rho_i}{\partial m_i}, \tag{7.62}$$

where $\rho_i = \sum_j m_j W_{ij}$, as per normal SPH. If we apply the derivative $\frac{\partial}{\partial m_i}$ to this term, and utilize the product rule again, we get

$$\frac{\partial \rho_i}{\partial m_i} = \sum_j \left(\frac{\partial m_j}{\partial m_i} W_{ij} + m_j \frac{\partial W_{ij}}{\partial m_i}\right), \tag{7.63}$$

where the first term is trivial as it is 0 for $i \neq j$ and 1 otherwise. This further simplifies the equation to

$$\frac{\partial \rho_i}{\partial m_i} = W_{ii} + \sum_j m_j \frac{\partial W_{ij}}{\partial m_i}. \tag{7.64}$$

If we now recall that $\frac{\partial W_{ij}}{\partial m_i}$, which was the actual partial derivative we were looking for, we get a recursive derivative with a dependency on all neighboring values as well. We can write this dependency as:

$$\frac{\partial W(r,h)}{\partial m_i} = \frac{\partial W(r,h)}{\partial h}\frac{\partial h}{\partial m_i}, \tag{7.65}$$

$$\frac{dh_i}{dm_i} = \frac{h_i}{3m_i} - \frac{h_i}{3\rho_i}\frac{\partial \rho_i}{\partial m_i}, \tag{7.66}$$

$$\frac{\partial \rho_i}{\partial m_i} = W_{ii} + \sum_j m_j \frac{\partial W(r,h)}{\partial m_i}. \tag{7.67}$$

Or, by simplifying the equations, we can simply write:

(7.68)
$$\frac{\partial W_{ij}}{\partial m_i} = \frac{\partial W_{ij}}{\partial h}\left[\frac{h_i}{3m_i} + \frac{h_i}{3\rho_i}\left(W_{ii} - \sum_k m_k \frac{\partial W_{ik}}{\partial m_i}\right)\right].$$

This could in theory be solved, approximately, by evaluating this term until the result converges within some ϵ threshold. However, this makes evaluating the gradient function prohibitively expensive.

We can, however, make an assumption by using non varying smoothing lengths where instead of using $h_i = \eta\sqrt[3]{\frac{m_i}{\rho_i}}$, we assume that the effective volume $V_i = \frac{m_i}{\rho_i}$ is always the same as the resting volume $V_i^0 = \frac{m_i}{\rho_i^0}$, where ρ_i^0 denotes the rest density of particle i, we get

(7.69)
$$h_i = \eta\sqrt[3]{\frac{m_i}{\rho_i^0}}.$$

This does not correct for varying amounts of mass in a spatial region, meaning that the constraint of $\rho_i V_i = \text{const}$, which is required for some CFD approaches, is violated, but this seems unavoidable and is common practice in computer animation. Doing this now changes $\frac{\partial h_i}{\partial m_i}$ significantly as $\frac{\partial \rho_i^0}{\partial m_i}$ becomes 0 and thus

(7.70)
$$\frac{dh_i}{dm_i} = \frac{h_i}{3m_i},$$

resulting in the final derivative of the kernel function by the mass of a particle as

(7.71)
$$\frac{\partial W_{ij}}{\partial m_i} = \frac{h_i}{3m_i}\frac{\partial W_{ij}}{\partial h_i},$$

which is comparatively trivial to calculate.

7.A.6 Derivative by mass ratio $\frac{\partial}{\partial \lambda_i} W_{ij}$

For later optimizations we have a set of weights $\Lambda_o = [\lambda_1, \dots, \lambda_n]$, which are constrained by

(7.72)
$$\sum_i \frac{1}{\lambda_i} = 1,$$

which means that we can determine an individual weight $\frac{1}{\lambda_j}$ in two ways. We can express the weight trivially as $\frac{1}{\lambda_j} = \frac{1}{\lambda_j}$ and using the other weights,by refactoring the constraint. These two expressions can be written as

(7.73)
$$\frac{1}{\lambda_j} = \frac{1}{\lambda_j} \wedge \frac{1}{\lambda_j} = 1 - \sum_{i \neq j} \frac{1}{\lambda_i}.$$

Applying the partial derivative to the first term is straight forward as we can simply write

$$\frac{\partial}{\partial \lambda_i}\left[\frac{1}{\lambda_j}\right] = \begin{cases} 0, & i \neq j \\ -\frac{1}{\lambda_i^2}, & \text{else} \end{cases} . \tag{7.74}$$

The other term is significantly more challenging as just directly applying the derivative would yield

$$\frac{\partial}{\partial \lambda_i}\left[1 - \sum_{i \neq j} \frac{1}{\lambda_i}\right] = \begin{cases} \frac{1}{\lambda_i^2}, & i \neq j \\ 0, & \text{else}, \end{cases} \tag{7.75}$$

which is not ideal as this is the equivalent of assuming that any change of ratio of one particle is applied equally to a single other particle, and every other particle assumes for it's derivative to take all of the change. To avoid this we instead assume that the change is distributed equally, which in turn yields

$$\frac{\partial}{\partial \lambda_i}\left[1 - \sum_{i \neq j} \frac{1}{\lambda_i}\right] = \begin{cases} \frac{1}{n-1}\frac{1}{\lambda_i^2}, & i \neq j \\ 0, & \text{else}, \end{cases} \tag{7.76}$$

with n being the size of the set of weights. Combined with the first formulation we could now combine these terms together to get a derivative applicable in every case. However, instead we introduce a matrix $\mathbb{M}$, which describes this distribution. This means that for the case of 1 to 1 we can define

$$\mathbb{M}_{ij} = \begin{cases} 1, & i \neq j \\ -1, & \text{else}, \end{cases} \tag{7.77}$$

and in the case of an even distribution

$$\mathbb{M}_{ij} = \begin{cases} \frac{1}{n-1}, & i \neq j \\ -1, & \text{else}. \end{cases} \tag{7.78}$$

A good distribution matrix $\mathbb{M}$ should fulfill the condition

$$\sum_j \mathbb{M}_{ij} = 1, \tag{7.79}$$

to preserve reasonable behavior. In general any arbitrary matrix could be chosen, i.e. $\mathbb{M} \in \mathbb{R}^{n \times n}$, but we only utilized the even distribution one. As such we can write:

$$\frac{\partial}{\partial \lambda_i}\left[\frac{1}{\lambda_j}\right] = \frac{\mathbb{M}_{ij}}{\lambda_i^2}. \tag{7.80}$$

This term complicates partial derivatives by λ_i significantly as, for example, $\frac{\partial W_{ns}}{\partial \lambda_i} \neq 0$ if s or n also relate to Λ_o, i.e. they are part of the same set of refined particles (this term will later be clarified) $\mathcal{S}_o$. Now applying the partial derivative to a kernel function we get:

$$\frac{\partial}{\partial \lambda_i}\left[W\left(|\mathbf{x_j}-\mathbf{x_s}|, \frac{\mathbf{h_s}+\mathbf{h_j}}{\mathbf{2}}\right)\right]. \tag{7.81}$$

Utilizing the chain rule, as before, we can write this as

$$\frac{\partial W_{sj}}{\partial r}\frac{\partial r}{\partial \lambda_i} + \frac{\partial W_{sj}}{\partial h}\frac{\partial h}{\partial \lambda_i}, \tag{7.82}$$

with $r = |\mathbf{x_j}-\mathbf{x_s}|$ and $h = \frac{h_s+h_j}{2}$. The first derivative term is zero, as $\frac{\partial r}{\partial \lambda_i} = 0$, but the second term remains. From before we know that

$$\frac{\partial W(r,h)}{\partial h} = -\frac{d}{h}W(r,h) - qW'(r,h), \tag{7.83}$$

which leaves only $\frac{\partial h}{\partial \lambda_i}$ as unknown. Due to symmetric support radii for interactions we can write

$$\frac{\partial h}{\partial \lambda_i} = \frac{1}{2}\frac{\partial h_s}{\partial \lambda_i} + \frac{1}{2}\frac{\partial h_j}{\partial \lambda_i}. \tag{7.84}$$

The derivatives of the support radii by the weights are only non zero if s or j are in the same set of refined particles $\mathcal{S}_o$ as i. As such we can rewrite this in terms of the indicator function

$$\mathbf{1}_{\mathcal{S}_o}(j) = \begin{cases} 1, & j \in \mathcal{S}_o \\ 0, & \text{else,} \end{cases} \tag{7.85}$$

as:

$$\frac{\partial h}{\partial \lambda_i} = \frac{1}{2}\left(\frac{\partial h_s}{\partial \lambda_i}\mathbf{1}_{\mathcal{S}_o}(s) + \frac{\partial h_j}{\partial \lambda_i}\mathbf{1}_{\mathcal{S}_o}(j)\right). \tag{7.86}$$

We can determine the support radius of a particle as

$$h_s = \eta\sqrt[3]{\frac{m_s}{\rho_s^0}} = \eta\sqrt[3]{\frac{1}{\lambda_s}\frac{m_o}{\rho_o^0}} = \eta\sqrt[3]{\frac{m_o}{\rho_o^0}}\sqrt[3]{\frac{1}{\lambda_s}} = \frac{h_o}{\sqrt[3]{\lambda_s}}. \tag{7.87}$$

If we now apply the partial derivative, with respect to the mass ratio of any particle i (which is part of the same set of refined particles as s) we first substitute $\frac{1}{\lambda_s} = c$ and get

(7.88)
$$h_s = \eta \sqrt[3]{\frac{m_s}{\rho_s^0}} = h_o \sqrt[3]{c},$$

where we can trivially apply $\frac{\partial}{\partial \lambda_i}$ as this is

(7.89)
$$\frac{\partial h_s}{\partial \lambda_i} = \frac{1}{3} h_o c^{-\frac{2}{3}} \frac{\partial c}{\partial \lambda_i} = \frac{1}{3} h_o \left(\frac{1}{\lambda_s}\right)^{-\frac{2}{3}} \frac{\partial \frac{1}{\lambda_s}}{\partial \lambda_i} = \frac{1}{3} h_o \frac{1}{\sqrt[3]{\lambda_s}} \lambda_s \frac{\partial \frac{1}{\lambda_s}}{\partial \lambda_i} = \frac{1}{3} h_s \lambda_s \frac{\mathbb{M}_{is}}{\lambda_i^2},$$

where the last step is done by resubstituting $h_s = \frac{h_o}{\sqrt[3]{\lambda_s}}$. Putting this back into $\frac{\partial h}{\partial \lambda_i}$ then yields

(7.90)
$$\frac{\partial h}{\partial \lambda_i} \frac{1}{2} \left(\frac{1}{3} h_s \lambda_s \frac{\mathbb{M}_{is}}{\lambda_i^2} \mathbf{1}_{\mathcal{S}_o}(s) + \frac{1}{3} h_j \lambda_j \frac{\mathbb{M}_{ij}}{\lambda_i^2} \mathbf{1}_{\mathcal{S}_o}(j) \right).$$

Collecting the common terms $\frac{1}{3\lambda_i^2}$ we can simplify this to

(7.91)
$$\frac{\partial h}{\partial \lambda_i} = \frac{1}{6\lambda_i^2} \left(h_s \lambda_s \mathbb{M}_{is} \mathbf{1}_{\mathcal{S}_o}(s) + h_j \lambda_j \mathbb{M}_{ij} \mathbf{1}_{\mathcal{S}_o}(j) \right),$$

which gives the final derivative term

(7.92)
$$\frac{\partial W_{sj}}{\partial \lambda_i} = \frac{1}{6\lambda_i^2} \left(h_s \lambda_s \mathbb{M}_{is} \mathbf{1}_{\mathcal{S}_o}(s) + h_j \lambda_j \mathbb{M}_{ij} \mathbf{1}_{\mathcal{S}_o}(j) \right) \frac{\partial W_{sj}}{\partial h_{sj}},$$

with

(7.93)
$$\frac{\partial W_{sj}}{\partial h_{sj}} = -\frac{1}{h_{sj}} \left(d W_{sj} + |\mathbf{x}_{sj}| W'_{sj} \right).$$

7.A.7 Summary

To sum it up we can define a kernel function in general for the interaction between two particles i and j as:

(7.94)
$$W_{ij} = W(|\mathbf{x_i} - \mathbf{x_j}|, \mathbf{h_{ij}}) = \frac{\alpha_\mathbf{d}}{\mathbf{h_{ij}^d}} \hat{\mathbf{W}} \left(\mathbf{q} = \frac{|\mathbf{x_i} - \mathbf{x_j}|}{\mathbf{h_{ij}}} \right)$$

with the helper notation

(7.95)
$$W'(r, h) = \frac{C}{h^d} \frac{\partial \hat{W}(q)}{\partial q}.$$

We can then get the following set of derivative terms:

(7.96)
$$\nabla_i W_{ij} = \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|} W'_{ij}.$$

(7.97) $$\frac{\partial W_{ij}}{\partial h_i} = -\frac{1}{2}\left[\frac{d}{h_{ij}}W_{ij} + qW'_{ij}\right]$$

(7.98) $$\frac{\partial W_{ij}}{\partial m_i} = \frac{h_i}{3m_i}\frac{\partial W_{ij}}{\partial h_i}$$

(7.99) $$\frac{\partial W_{sj}}{\partial h_{sj}} = -\frac{1}{h_{sj}}\left(dW_{sj} + |\mathbf{x}_{sj}|W'_{sj}\right)$$

(7.100) $$\frac{\partial W_{sj}}{\partial \lambda_i} = \frac{1}{6\lambda_i^2}\left(h_s\lambda_s\mathbb{M}_{is}\mathbf{1}_{\mathcal{S}_o}(s) + h_j\lambda_j\mathbb{M}_{ij}\mathbf{1}_{\mathcal{S}_o}(j)\right)\frac{\partial W_{sj}}{\partial h_{sj}}, \mathbb{M}_{ij} = \begin{cases}\frac{1}{n-1}, & i \neq j\\ -1, & \text{else,}\end{cases}$$

7.A.8 Refinement error terms

In our contribution we utilized a generic error formulation, which has been used before, and before calculating required derivative terms we would like to expand on this error term and it's motivation and aspects.

In general there are five quantities that should be preserved:

1. Conservation of Mass
2. Conservation of Linear Momentum
3. Conservation of Angular Momentum
4. Conservation of Energy
5. Conservation of Field Quantities

Out of these conservation of mass is the easiest quantity to preserve as we can simply enforce the constraint

(7.101) $$\sum_{s\in\mathcal{S}_o} m_s = m_o.$$

Conservation of Linear momentum is also straight forward as we can set the velocity of all refined particles equal to the initial particle

(7.102) $$\mathbf{v_s} = \mathbf{v_o}, \forall \mathrm{s} \in \mathcal{S}_\mathbf{o},$$

which under mass conservation guarantees that

(7.103) $$\sum_{s\in\mathcal{S}_o} m_s\mathbf{v_s} = \mathbf{m_o}\mathbf{v_o}.$$

Similarly Angular Momentum can be conserved straight forward if the newly refined particles have equal velocity and have a center of mass that is equal to the position of the original particle, i.e. by using symmetric refinement patterns. However, enforcing this quantity exactly is very difficult and as such is usually not explicitly taken care of.

This leaves conservation of energy and conservation of field quantities and prior work has shown that it is not mathematically possible to conserve both quantities at the same time. Obviously, decreasing or even increasing energy in a simulation is not ideal but neither is a non-conservation of field quantities.

As everything in SPH is based around the density, keeping the density field conserved across a refinement process might be the best option as locally increasing density would lead to compression, which incompressible SPH methods tend to not handle very well (as they are designed to prevent errors from occurring not necessarily to remove them after they suddenly appear). As such we chose to prioritize conservation of field quantities (specifically density) over conservation of energy.

Using this conservation approach for an adaptive process we can determine an error at any position in space due to a refinement process by defining a spatial error

$$\tau(\mathbf{x}) = \rho^*(\mathbf{x}) - \rho(\mathbf{x}) \tag{7.104}$$

as the difference of density before (ρ) and after (ρ^*) refinement. This error term can be evaluated across a domain, which due to compact support radii is bounded by the h_o, as

$$\mathcal{E} = \int_\Omega \tau(\mathbf{x})^\mathbf{2}\mathbf{dx}, \tag{7.105}$$

using a sum of squared error term.

This is a continuous error function, which could also be interpreted as enforcing $\frac{D\rho}{Dt}$ (divergence-freedom) over a refinement step, instead of a time step.

We can modify τ by realizing that the difference of before and after can be represented as removing the original particle and adding the particles, keeping everything else equal as

$$\begin{aligned}\tau(\mathbf{x}) = &\underbrace{\sum_{j\in\mathcal{N}_\mathbf{x}\backslash o} m_j W\left(|\mathbf{x}-\mathbf{x_j}|,\mathbf{h}\right) + \sum_{s\in\mathcal{S}_o} m_s W\left(|\mathbf{x}-\mathbf{x_s}|,\mathbf{h}\right)}_{\rho^*(\mathbf{x})}\\ &- \left[\underbrace{\sum_{j\in\mathcal{N}_\mathbf{x}\backslash o} m_j W\left(|\mathbf{x}-\mathbf{x_j}|,\mathbf{h}\right) + m_o W\left(|\mathbf{x}-\mathbf{x_o}|,\mathbf{h}\right)}_{\rho^*(\mathbf{x})}\right]\end{aligned} \tag{7.106}$$

simplifies to:

$$\tau(\mathbf{x}) = \sum_{\mathbf{s}\in\mathcal{S}_\mathbf{o}} \mathbf{m_s W}\left(|\mathbf{x}-\mathbf{x_s}|,\mathbf{h}\right) - \mathbf{m_o W}\left(|\mathbf{x}-\mathbf{x_o}|,\mathbf{h}\right). \tag{7.107}$$

However, this term cannot be evaluated easily for symmetric SPH formulations due to the h term. For a scatter based formulation, as it was used for weakly compressible methods in the past, h simply evaluates to h_j, i.e. the support radius of the interacting particle. This would change the τ term to now be

$$\tau(\mathbf{x}) = \sum_{\mathbf{s}\in\mathcal{S}_\mathbf{o}} \mathbf{m_s W}\left(|\mathbf{x}-\mathbf{x_s}|,\mathbf{h_s}\right) - \mathbf{m_o W}\left(|\mathbf{x}-\mathbf{x_o}|,\mathbf{h_o}\right), \tag{7.108}$$

which is straight forward to compute as both h_s and h_o are either easy to calculate or known (see the prior discussion on kernel functions). For symmetric formulations however h evaluates to $\frac{h(\mathbf{x})+h_j}{2}$, where $h(\mathbf{x})$ represents the support radius for a location. This could potentially be evaluate using

$$h(\mathbf{x}) = \eta\sqrt[3]{\frac{m(\mathbf{x})}{\rho(\mathbf{x})}}, \rho(\mathbf{x}) = \sum_{j\in\mathcal{N}_\mathbf{x}} m_j W\left(|\mathbf{x}-\mathbf{x_s}|, \frac{\mathbf{h(x)-h_j}}{\mathbf{2}}\right). \tag{7.109}$$

This term however is not directly computable as we have a dependence of $h(\mathbf{x})$ to itself, where we would iteratively solve these equations until the result convergence, and a dependence on the mass of a point $m(\mathbf{x})$, which would be resolved by using some trapezoidal integration scheme, which assigns a volume to each integration point and thus a mass. This complexity makes this error term computationally very expensive and not useful in practice due to finite resources.

We resolve this problem by realizing that the core issue stems from requiring $h(\mathbf{x})$, which is unknown. However, the support radius is known (exactly) at the positions of neighbors of the original particle, the position of the original particle and can be evaluated at the positions of the refined particles. Assuming the common practice of computer animation that $V = V^0$ (for calculations of support radii), $h_s = \eta\sqrt[3]{V_s^0}$ and $m_s = \frac{1}{\lambda_s}m_o$ we can determine the error on the positions of the neighbors of the original particle as

$$\tau(\mathbf{x_n}) = \tau_\mathbf{n} = \mathbf{m_o}\sum_{\mathbf{s}\in\mathcal{S}_\mathbf{o}} \frac{\mathbf{1}}{\lambda_\mathbf{s}}\mathbf{W_{ns}} - \mathbf{m_o W_{no}}, \forall\mathbf{n}\in\bar{\mathcal{N}_\mathbf{o}} \tag{7.110}$$

and on the positions of the refined particles as

$$\tau(\mathbf{x_s}) = \tau_\mathbf{s} = \sum_{\mathbf{j}\in\mathcal{N}_\mathbf{s}} \mathbf{m_j W_{sj}} + \mathbf{m_o}\sum_{\mathbf{k}\in\mathcal{S}_\mathbf{o}} \frac{\mathbf{1}}{\lambda_\mathbf{k}}\mathbf{W_{sk}} - \rho_\mathbf{o}, \forall\mathbf{s}\in\mathcal{S}_\mathbf{o}. \tag{7.111}$$

In the second term we assume that $\rho(\mathbf{x_s}) = \rho(\mathbf{x_o})$, as this conserves the quantity on the refined particles with respect to the original particle and as $\mathcal{N}_s \subseteq \mathcal{N}_o$ we cannot make the simplification that was applied to the error term on the position of neighboring particles.

In order to calculate the overall error terms we then simply weigh these terms by the mass of the respective particle and get the following terms:

$$\mathbb{E}_n = \sum_{n \in \bar{\mathcal{N}}_o} m_n \tau_n^2, \mathbb{E}_s = \sum_{s \in \mathcal{S}_o} m_s \tau_s^2, \tag{7.112}$$

$$\mathbb{E} = \sum_{n \in \bar{\mathcal{N}}_o} m_n \tau_n^2 + \sum_{s \in \mathcal{S}_o} m_s \tau_s^2. \tag{7.113}$$

As this error describes the violation of the conservation of density (as a field quantity) through refinement, a minimal error will mean the minimal violation, i.e. we get an optimization problem of $min\mathbb{E}$.

To recap:

In prior work Feldmann used the following set of error terms (for scatter based SPH):

$$\mathcal{E}^{\mathsf{F}} = \int_\Omega \tau^{\mathsf{F}}(\mathbf{x})\mathbf{dx}, \tau^{\mathsf{F}}(\mathbf{x}) = \sum_{\mathbf{s} \in \mathcal{S}_\mathbf{o}} \mathbf{m_s W}\left(|\mathbf{x} - \mathbf{x_s}|, \mathbf{h_s}\right) - \mathbf{m_o W}\left(|\mathbf{x} - \mathbf{x_o}|, \mathbf{h_o}\right), \tag{7.114}$$

with $\int_\Omega$ being discretized using a trapezoidal integration scheme.

We could define the following set of error terms:

$$\begin{aligned} \mathcal{E}^{\mathsf{S}} &= \int_\Omega \tau^{\mathsf{S}}(\mathbf{x})\mathbf{dx}, \tau^{\mathsf{S}}(\mathbf{x}) \\ &= \sum_{s \in \mathcal{S}_o} m_s W\left(|\mathbf{x} - \mathbf{x_s}|, \frac{\mathbf{h(x)} + \mathbf{h_s}}{\mathbf{2}}\right) - m_o W\left(|\mathbf{x} - \mathbf{x_o}|, \frac{\mathbf{h(x)} + \mathbf{h_s}}{\mathbf{2}}\right), \end{aligned} \tag{7.115}$$

with $\int_\Omega$ being discretized using a trapezoidal integration scheme and $h(\mathbf{x})$ being iteratively solved until the value converges.

Finally we can define the following set of error terms:

$$\mathbb{E} = \sum_{n \in \bar{\mathcal{N}}_o} m_n \tau_n^2 + \sum_{s \in \mathcal{S}_o} m_s \tau_s^2, \tag{7.116}$$

$$\tau_n = m_o \sum_{s \in \mathcal{S}} \frac{1}{\lambda_s} W_{ns} - m_o W_{no}, \tau_s = \sum_{j \in \mathcal{N}_s} m_j W_{sj} + m_o \sum_{k \in \mathcal{S}_o} \frac{1}{\lambda_k} W_{sk} - \rho_o, \tag{7.117}$$

where $\mathbb{E}$ is already discretized as described previously.

In order to conserve linear momentum and mass, we cannot modify all aspects of the refined particles. As such we can only modify their relative positions ($\mathbf{x}_\mathcal{S} : \mathbf{x_s}, \mathrm{s} \in \mathcal{S}_\mathbf{o}$) and their relative mass distribution ($m_\mathcal{S} : m_s = \frac{1}{\lambda_s} m_o, s \in \mathcal{S}_o$) under constraints ($\lambda_s > 0 \forall s \in \mathcal{S}_o$, $\sum_{s \in \mathcal{S}_o} m_s = m_o$).

If we were to define an optimization problem directly

(7.118) $$\text{min}_{\mathbf{x}_\mathcal{S},\lambda_\mathcal{S}}\mathcal{E} = \text{min}_{\mathbf{x}_\mathcal{S},\lambda_\mathcal{S}} \int_\Omega \tau(\mathbf{x})\mathbf{dx},$$

we would find an optimal but useless solution, regardless of the specific τ used as the solution:

(7.119) $$\mathbf{x}_{\mathbf{s_1}} = \mathbf{x}_\mathbf{o}, \mathbf{m}_{\mathbf{s_1}} = \mathbf{m}_\mathbf{o}, \mathbf{m}_{\mathbf{s}\in\mathcal{S}\setminus\mathbf{s_1}} = \epsilon$$

is ideal but useless as it sets the mass of all but one particle to virtually zero and makes one particle identical to the original particle. To avoid such solutions prior work has split the optimization of position and mass to be in sequence, which we also do for our optimization method. As such we first optimize

(7.120) $$\text{min}_{\mathbf{x}_\mathcal{S}}\mathbb{E} = \text{min}_{\mathbf{x}_\mathcal{S}}\left[\sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2\right]$$

and then optimize

(7.121) $$\text{min}_{\lambda_\mathcal{S}}\mathbb{E} = \text{min}_{\lambda_\mathcal{S}}\left[\sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2\right]$$

under the constraints $\lambda_s > 0 \forall s \in \mathcal{S}_o$ and $\sum_{s\in\mathcal{S}_o} m_s = m_o$. This split also allows us to use an unconstrained optimization method for the positional optimization (which involves d times as many parameters) and only requires a constrained optimization for the mass distribution optimization.

7.A.9 Optimizing by position

In general we had two distinct error terms in

(7.122) $$\mathbb{E} = \sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2,$$

where one represents the error evaluated on the positions of the neighbors of the original particle

(7.123) $$\tau_n = \sum_{s\in\mathcal{S}} m_s W_{ns} - m_o W_{no}$$

and one represents the error evaluated on the positions of the newly inserted particle

(7.124) $$\tau_s = \sum_{j\in\mathcal{N}_s} m_j W_{sj} + \sum_{k\in\mathcal{S}_o} m_k W_{sk} - \rho_o.$$

As the optimization with respect to position, i.e.

$$\text{min}_{\mathbf{x}_\mathcal{S}}\mathbb{E} = \text{min}_{\mathbf{x}_\mathcal{S}}\left[\sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2\right], \tag{7.125}$$

is an unconstrained optimization problem, we don't need any constraints and can simply utilize any optimization method. Common choices could include a gradient descent (as is done for our online optimization) and L-BFGS-B (as is done for our a-priori optimization), but most methods require at least the first derivative of the error function with respect to the optimization variables.

As such, if we apply the gradient with respect to the newly refined particle i, $\nabla_i = \left[\frac{\partial}{\partial x_i}, \frac{\partial}{\partial y_i}, \frac{\partial}{\partial z_i}\right]$, to the discretized error function we can simply move the gradient operator into the summation as

$$\nabla_i\mathbb{E} = \sum_{s\in\mathcal{S}_o} 2m_s\tau_s\nabla_i\tau_s + \sum_{n\in\bar{\mathcal{N}}_o} 2m_n\tau_n\nabla_i\tau_n, \tag{7.126}$$

which can be further simplified by splitting the first summation based on i as

$$\nabla_i\mathbb{E} = 2m_i\tau_i\nabla_i\tau_i + \sum_{s\in\mathcal{S}_o\setminus i} 2m_s\tau_s\nabla_i\tau_s + \sum_{n\in\bar{\mathcal{N}}_o} 2m_n\tau_n\nabla_i\tau_n. \tag{7.127}$$

Here we now only require the terms $\nabla_i\tau_i$, $\nabla_i\tau_s$ and $\nabla_i\tau_n$ as everything else has been discussed before.

7.A.10 For refined particles $\nabla_i\tau_i$

Applying ∇_i to τ_i directly results in

$$\begin{aligned}\nabla_i\tau_i &= \nabla_i\left[\sum_{j\in\mathcal{N}_i} m_jW_{ij} + \sum_{k\in\mathcal{S}_o} m_kW_{ik} - \rho_o\right]\\ &= \underbrace{\sum_{j\in\mathcal{N}_i} m_j\nabla_iW_{ij}}_{\text{I}} + \underbrace{\sum_{k\in\mathcal{S}_o} m_k\nabla_iW_{ik}}_{\text{II}} - \underbrace{\nabla_i\rho_o}_{\text{III}}.\end{aligned} \tag{7.128}$$

I and II can be evaluated as is and cannot be further simplified. III however evaluates to 0 as the density of the original particle does not depend, in any way, on the refined particles position. This means that we get the following term:

$$\nabla_i\tau_i = \sum_{j\in\mathcal{N}_i} m_j\nabla_iW_{ij} + \sum_{k\in\mathcal{S}_o} m_k\nabla_iW_{ik}. \tag{7.129}$$

7.A.11 For refined particles $\nabla_i \tau_s$

Applying ∇_i to τ_s directly results in

(7.130)
$$\begin{aligned}\nabla_i \tau_s &= \nabla_i \left[\sum_{j\in\mathcal{N}_s} m_j W_{sj} + \sum_{k\in\mathcal{S}_o} m_k W_{sk} - \rho_o\right] \\ &= \underbrace{\sum_{j\in\mathcal{N}_s} m_j \nabla_i W_{sj}}_{\text{I}} + \underbrace{\sum_{k\in\mathcal{S}_o} m_k \nabla_i W_{sk}}_{\text{II}} - \underbrace{\nabla_i \rho_o}_{\text{III}}.\end{aligned}$$

Where again III evaluates to 0. I evaluates to 0 as well as s cannot be i, due to the separation of summations before, and as such there is no term in this summation that can contain i. II similarly evaluates to 0 for all $k \neq i$, which leaves yields

(7.131)
$$m_i \nabla_i W_{si}.$$

Due to the symmetry of a kernel function we can then also write

(7.132)
$$\nabla_i \tau_s = m_i \nabla_i W_{is}.$$

7.A.12 For neighboring particles $\nabla_i \tau_n$

Applying ∇_i to τ_n directly results in

(7.133)
$$\nabla_i \tau_n = \nabla_i \left[\sum_{s\in\mathcal{S}} m_s W_{ns} - m_o W_{no}\right] = \underbrace{\sum_{s\in\mathcal{S}} m_s \nabla_i W_{ns}}_{\text{I}} - \underbrace{m_o \nabla_i W_{no}}_{\text{II}}.$$

Similar to before II evaluates to 0. In the summation term I the only term that remains is

(7.134)
$$m_i \nabla_i W_{ni}.$$

Due to symmetry of the kernel function we can then write

(7.135)
$$\nabla_i \tau_n = m_i \nabla_i W_{in}.$$

7.A.13 Complete derivative term

Combining the previous terms we can then write the objective function as:

(7.136)
$$\mathbb{E} = \sum_{n\in\tilde{\mathcal{N}}_o} m_n \tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s \tau_s^2$$

and the gradient of this objective function with respect to the position of a refined particle ∇_i as

$$\begin{aligned}\nabla_i \mathbb{E} =& 2m_i\tau_i \left[\sum_{j\in\mathcal{N}_i} m_j \nabla_i W_{ij} + \sum_{k\in\mathcal{S}_o} m_k \nabla_i W_{ik}\right] \\ &+ \sum_{s\in\mathcal{S}_o\setminus i} 2m_s\tau_s m_i \nabla_i W_{is} + \sum_{n\in\bar{\mathcal{N}}_o} 2m_n\tau_n m_i \nabla_i W_{in},\end{aligned} \tag{7.137}$$

which can be refactored slightly to collect similar terms:

$$\begin{aligned}\nabla_i \mathbb{E} = 2m_i[&\tau_i \sum_{j\in\mathcal{N}_i} m_j \nabla_i W_{ij} + \tau_i \sum_{k\in\mathcal{S}_o} m_k \nabla_i W_{ik} \\ &+ \sum_{s\in\mathcal{S}_o\setminus i} \tau_s m_s \nabla_i W_{is} + \sum_{n\in\bar{\mathcal{N}}_o} \tau_n m_n \nabla_i W_{in}].\end{aligned} \tag{7.138}$$

As $\nabla_i W_{ii} = 0$ and by changing the variable naming for the second loop we can further simplify this as:

$$\nabla_i \mathbb{E} = 2m_i \left[\sum_{j\in\mathcal{N}_i} \tau_i m_j \nabla_i W_{ij} + \sum_{n\in\bar{\mathcal{N}}_o} \tau_n m_n \nabla_i W_{in} + \sum_{s\in\mathcal{S}_o} m_s \left(\tau_i - \tau_s\right) \nabla_i W_{is}\right]. \tag{7.139}$$

Furthermore as $\mathcal{N}_i \setminus i \subseteq \bar{\mathcal{N}}_o$ with $\nabla_i W_{in} = \forall n \in \bar{\mathcal{N}}_o \setminus \mathcal{N}_i$ and $\nabla_i W_{ii} = 0$, we can further simplify as

$$\nabla_i \mathbb{E} = 2m_i \left[\underbrace{\sum_{n\in\bar{\mathcal{N}}_o} m_n \left(\tau_i + \tau_n\right) \nabla_i W_{in}}_{\nabla_i \mathbb{E}_{\mathcal{N}}} + \underbrace{\sum_{s\in\mathcal{S}_o} m_s \left(\tau_i + \tau_s\right) \nabla_i W_{is}}_{\nabla_i \mathbb{E}_{\mathcal{S}}}\right], \tag{7.140}$$

where we have two summations over independent sets of neighbors and refined particles, which could be referred to as $\nabla_i \mathbb{E}_{\mathcal{N}}$ and $\nabla_i \mathbb{E}_{\mathcal{S}}$. This accordingly yields

$$\nabla_i \mathbb{E} = 2m_i \left[\nabla_i \mathbb{E}_{\mathcal{N}} + \nabla_i \mathbb{E}_{\mathcal{S}}\right]. \tag{7.141}$$

$$\nabla_i \mathbb{E}_{\mathcal{N}} = \sum_{n\in\bar{\mathcal{N}}_o} m_n \left(\tau_i + \tau_n\right) \nabla_i W_{in}, \nabla_i \mathbb{E}_{\mathcal{S}} = \sum_{s\in\mathcal{S}_o} m_s \left(\tau_i + \tau_s\right) \nabla_i W_{is}. \tag{7.142}$$

7.A.14 Optimizing by mass

Optimizing the mass directly would be possible, i.e. optimizing $m_{\mathcal{S}}$ but this is not ideal. Certain methods (i.e. SLSQP) can enforce constraints of the form $\sum_n 1/x_n = c$ easier than a constraint of the form $\sum_n x_n = c$. As such we first replace the mass of a refined particle s, i.e. m_s, with a ratio of the original mass

$$m_s = \frac{1}{\lambda_s} m_o, \tag{7.143}$$

under the constraint $\sum_{s \in \mathcal{S}_o} 1/\lambda_s = 1$. Applying this to the discretized error function yields

$$\mathbb{E} = \sum_{n \in \bar{\mathcal{N}}_o} m_n \tau_n^2 + m_o \sum_{s \in \mathcal{S}_o} \frac{1}{\lambda_s} \tau_s^2, \tag{7.144}$$

and further applying it to the error terms τ_n and τ_s gives:

$$\tau_n = m_o \left(\sum_{s \in \mathcal{S}} \frac{1}{\lambda_s} W_{ns} - W_{no} \right), \tau_s = \sum_{j \in \mathcal{N}_s} m_j W_{sj} + m_o \sum_{k \in \mathcal{S}_o} \frac{1}{\lambda_k} W_{sk} - \rho_o. \tag{7.145}$$

A set of weights $[\lambda_1, ...\lambda_n]$ corresponding to a set of refined particles $\mathcal{S}_o$ is denoted by Λ_o for readability. This gives the optimization problem

$$\text{min}_{\Lambda_\mathcal{S}} \mathbb{E} = \text{min}_{\Lambda_\mathcal{S}} \left[\sum_{n \in \bar{\mathcal{N}}_o} m_n \tau_n^2 + m_o \sum_{s \in \mathcal{S}_o} \frac{1}{\lambda_s} \tau_s^2 \right]. \tag{7.146}$$

As this is a constrained problem, many common optimization methods cannot be applied here, i.e. gradient descent, but some methods still require a derivative of the error function with respect to a variable and as such these are also given here.

As such, if we apply the derivative with respect to the newly refined particle i's mass ratio, $\frac{\partial}{\partial \lambda_i}$, to the discretized error function we get

$$\frac{\partial \mathbb{E}}{\partial \lambda_i} = \sum_{n \in \bar{\mathcal{N}}_o} 2 m_n \tau_n \frac{\partial \tau_n}{\partial \lambda_i} + \frac{\partial}{\partial \lambda_i} \left[m_o \sum_{s \in \mathcal{S}_o} \frac{1}{\lambda_s} \tau_s^2 \right], \tag{7.147}$$

This then gives us:

$$\frac{\partial \mathbb{E}}{\partial \lambda_i} = \sum_{n \in \bar{\mathcal{N}}_o} 2 m_n \tau_n \frac{\partial \tau_n}{\partial \lambda_i} + m_o \sum_{s \in \mathcal{S}_o} \left(\frac{\partial \frac{1}{\lambda_s}}{\partial \lambda_i} \tau_s^2 + \frac{2\tau_s}{\lambda_s} \frac{\partial \tau_s}{\partial \lambda_i} \right) \tag{7.148}$$

Recall that

$$\frac{\partial}{\partial \lambda_i} \left[\frac{1}{\lambda_j} \right] = \frac{\mathbb{M}_{ij}}{\lambda_i^2}. \tag{7.149}$$

Going back to the partial derivative of $\mathbb{E}$ we can then write:

$$\frac{\partial \mathbb{E}}{\partial \lambda_i} = \sum_{n \in \bar{\mathcal{N}}_o} 2 m_n \tau_n \frac{\partial \tau_n}{\partial \lambda_i} + m_o \sum_{s \in \mathcal{S}_o} \frac{2\tau_s}{\lambda_s} \frac{\partial \tau_s}{\partial \lambda_i} + m_o \sum_{s \in \mathcal{S}_o} \frac{\mathbb{M}_{ij}}{\lambda_i^2} \tau_s^2, \tag{7.150}$$

where two partial derivatives, $\frac{\partial \tau_n}{\partial \lambda_i}$ and $\frac{\partial \tau_s}{\partial \lambda_i}$, remain similar to the positional derivatives of $\nabla_i \tau_n$ and $\nabla_i \tau_s$. However, note that the term $\nabla_i \tau_i$ does not appear directly as this was only a reasonable simplification for spatial derivatives, as $\nabla_i W_{ii} = 0$, which is not the case here.

7.A.15 For neighboring particles $\frac{\partial \tau_n}{\partial \lambda_i}$

First, recall the definition of τ_n using the weights Λ_o as

$$\tau_n = m_o \left(\sum_{s \in \mathcal{S}} \frac{1}{\lambda_s} W_{ns} - W_{no} \right). \tag{7.151}$$

Applying the partial derivative $\frac{\partial}{\partial \lambda_i}$ to this term, and pulling the derivative into the braces, gives

$$\frac{\partial \tau_n}{\partial \lambda_i} = m_o \left(\frac{\partial}{\partial \lambda_i} \left[\sum_{s \in \mathcal{S}} \frac{1}{\lambda_s} W_{ns} \right] - \frac{\partial W_{no}}{\partial \lambda_i} \right), \tag{7.152}$$

where $\partial W_{no} / \partial \lambda_i$ is independent of i and can be dropped. This leaves the remaining term

$$\frac{\partial \tau_n}{\partial \lambda_i} = m_o \frac{\partial}{\partial \lambda_i} \left[\sum_{s \in \mathcal{S}} \frac{1}{\lambda_s} W_{ns} \right], \tag{7.153}$$

where we can move the derivative into the summation and use the product rule to get

$$\frac{\partial \tau_n}{\partial \lambda_i} = m_o \sum_{s \in \mathcal{S}} \left(\frac{\partial}{\partial \lambda_i} \left[\frac{1}{\lambda_s} \right] W_{ns} + \frac{1}{\lambda_s} \frac{\partial}{\partial \lambda_i} \left[W_{ns} \right] \right), \tag{7.154}$$

which can be further expanded into

$$\frac{\partial \tau_n}{\partial \lambda_i} = m_o \sum_{s \in \mathcal{S}_o} \left(\frac{\mathbb{M}_{is}}{\lambda_i^2} W_{ns} + \frac{1}{\lambda_s} \frac{\partial W_{ns}}{\partial \lambda_i} \right). \tag{7.155}$$

7.A.16 For refined particles $\frac{\partial \tau_s}{\partial \lambda_i}$

First, recall the definition of τ_s using the weights Λ_o as

$$\tau_s = \sum_{j \in \mathcal{N}_s} m_j W_{sj} + m_o \sum_{k \in \mathcal{S}_o} \frac{1}{\lambda_k} W_{sk} - \rho_o. \tag{7.156}$$

If we now apply the partial derivative $\frac{\partial}{\partial \lambda_i}$ to this we get

$$\frac{\partial \tau_s}{\partial \lambda_i} = \sum_{j \in \mathcal{N}_s} m_j \frac{\partial W_{sj}}{\partial \lambda_i} + m_o \sum_{k \in \mathcal{S}_o} \frac{\partial}{\partial \lambda_i} \left[\frac{1}{\lambda_k} W_{sk} \right] - \frac{\partial \rho_o}{\partial \lambda_i}, \tag{7.157}$$

where the last term is independent of i and thus 0. We next utilize the product rule on the second term, which yields

$$\frac{\partial \tau_s}{\partial \lambda_i} = \sum_{j \in \mathcal{N}_s} m_j \frac{\partial W_{sj}}{\partial \lambda_i} + m_o \sum_{k \in \mathcal{S}_o} \left(\frac{\partial}{\partial \lambda_i} \left[\frac{1}{\lambda_k} \right] W_{sk} + \frac{1}{\lambda_k} \frac{\partial}{\partial \lambda_i} \left[W_{sk} \right] \right), \tag{7.158}$$

which can be further simplified into

$$\frac{\partial \tau_s}{\partial \lambda_i} = \sum_{j \in \mathcal{N}_s} m_j \frac{\partial W_{sj}}{\partial \lambda_i} + m_o \sum_{k \in \mathcal{S}_o} \left(\frac{\mathbb{M}_{ik}}{\lambda_i^2} W_{sk} + \frac{1}{\lambda_k} \frac{\partial W_{sk}}{\partial \lambda_i} \right). \tag{7.159}$$

7.A.17 Putting it together

First recall the initial formulation:

$$\frac{\partial \mathbb{E}}{\partial \lambda_i} = \sum_{n \in \bar{\mathcal{N}}_o} 2 m_n \tau_n \frac{\partial \tau_n}{\partial \lambda_i} + m_o \sum_{s \in \mathcal{S}_o} \frac{2\tau_s}{\lambda_s} \frac{\partial \tau_s}{\partial \lambda_i} + m_o \sum_{s \in \mathcal{S}_o} \frac{\mathbb{M}_{is}}{\lambda_i^2} \tau_s^2, \tag{7.160}$$

we can insert the prior summations:

$$\begin{aligned} \frac{\partial \mathbb{E}}{\partial \lambda_i} = & \sum_{n \in \bar{\mathcal{N}}_o} 2 m_n \tau_n \left[m_o \sum_{s \in \mathcal{S}_o} \left(\frac{\mathbb{M}_{is}}{\lambda_i^2} W_{ns} + \frac{1}{\lambda_s} \frac{\partial W_{ns}}{\partial \lambda_i} \right) \right] + \\ & m_o \sum_{s \in \mathcal{S}_o} \frac{2\tau_s}{\lambda_s} \left[\sum_{j \in \mathcal{N}_s} m_j \frac{\partial W_{sj}}{\partial \lambda_i} + m_o \sum_{k \in \mathcal{S}_o} \left(\frac{\mathbb{M}_{ik}}{\lambda_i^2} W_{sk} + \frac{1}{\lambda_k} \frac{\partial W_{sk}}{\partial \lambda_i} \right) \right] + \\ & m_o \sum_{s \in \mathcal{S}_o} \frac{\mathbb{M}_{is}}{\lambda_i^2} \tau_s^2. \end{aligned} \tag{7.161}$$

In order to simplify this term we first split every summation term apart and rename j to n and, as $\mathcal{N}_s \subseteq \bar{\mathcal{N}}_o$, we also update the set affinity:

$$\begin{aligned}\frac{\partial \mathbb{E}}{\partial \lambda_i} =& \sum_{s\in\mathcal{S}_o}\sum_{n\in\bar{\mathcal{N}}_o} 2m_n\tau_n m_o \left[\frac{\mathbb{M}_{is}}{\lambda_i^2}W_{ns} + \frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial \lambda_i}\right] + \\ &\sum_{s\in\mathcal{S}_o}\sum_{n\in\bar{\mathcal{N}}_o} \frac{2\tau_s}{\lambda_s} m_o m_n \left[\frac{\partial W_{sn}}{\partial \lambda_i}\right] + \\ &\sum_{s\in\mathcal{S}_o}\sum_{k\in\mathcal{S}_o} \frac{2\tau_s}{\lambda_s} m_o^2 \left[\frac{\mathbb{M}_{ik}}{\lambda_i^2}W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial \lambda_i}\right] + \\ &\sum_{s\in\mathcal{S}_o} m_o \frac{\mathbb{M}_{is}}{\lambda_i^2}\tau_s^2.\end{aligned} \tag{7.162}$$

Collecting summations we can simplify:

$$\begin{aligned}\frac{\partial \mathbb{E}}{\partial \lambda_i} =& \sum_{s\in\mathcal{S}_o}\Bigg(2m_o \sum_{n\in\bar{\mathcal{N}}_o} m_n \left[\tau_n\frac{\mathbb{M}_{is}}{\lambda_i^2}W_{ns} + \tau_n\frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial \lambda_i} + \tau_s\frac{1}{\lambda_s}\frac{\partial W_{sn}}{\partial \lambda_i}\right] + \\ &\sum_{k\in\mathcal{S}_o} \frac{2\tau_s}{\lambda_s} m_o^2 \left[\frac{\mathbb{M}_{ik}}{\lambda_i^2}W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial \lambda_i}\right] + m_o\frac{\mathbb{M}_{is}}{\lambda_i^2}\tau_s^2\Bigg),\end{aligned} \tag{7.163}$$

which due to symmetry further simplifies

$$\begin{aligned}\frac{\partial \mathbb{E}}{\partial \lambda_i} =& \sum_{s\in\mathcal{S}_o}\big(2m_o \sum_{n\in\bar{\mathcal{N}}_o} m_n \left[\tau_n\frac{\mathbb{M}_{is}}{\lambda_i^2}W_{ns} + \frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial \lambda_i}(\tau_n+\tau_s)\right] + \\ &2m_o\sum_{k\in\mathcal{S}_o} m_s \left[\tau_s\frac{\mathbb{M}_{ik}}{\lambda_i^2}W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial \lambda_i}\tau_s\right] + m_o\frac{\mathbb{M}_{is}}{\lambda_i^2}\tau_s^2\big),\end{aligned} \tag{7.164}$$

or by substitution:

$$\frac{\partial \mathbb{E}}{\partial \lambda_i} = \sum_{s\in\mathcal{S}_o}\left(2m_o\frac{\partial \mathbb{E}^s_{\mathcal{N}}}{\partial \lambda_i} + 2m_o\frac{\partial \mathbb{E}^s_{\mathcal{S}}}{\partial \lambda_i} + m_o\frac{\mathbb{M}_{is}}{\lambda_i^2}\right), \tag{7.165}$$

with

$$\begin{aligned}\frac{\partial \mathbb{E}^s_{\mathcal{N}}}{\partial \lambda_i} =& \sum_{n\in\bar{\mathcal{N}}_o} m_n \left[\tau_n\frac{\mathbb{M}_{is}}{\lambda_i^2}W_{ns} + \frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial \lambda_i}(\tau_n+\tau_s)\right], \frac{\partial \mathbb{E}^s_{\mathcal{S}}}{\partial \lambda_i} \\ =& \sum_{k\in\mathcal{S}_o} m_s \left[\tau_s\frac{\mathbb{M}_{ik}}{\lambda_i^2}W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial \lambda_i}\tau_s\right].\end{aligned} \tag{7.166}$$

7.A.18 The optimization problem

The overall problems initially described were

(7.167)
$$\begin{aligned}\text{min}_{\mathbf{x}_{\mathcal{S}}}\mathbb{E} &= \text{min}_{\mathbf{x}_{\mathcal{S}}}\left[\sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2\right],\\ \text{min}_{\lambda_{\mathcal{S}}}\mathbb{E} &= \text{min}_{\lambda_{\mathcal{S}}}\left[\sum_{n\in\bar{\mathcal{N}}_o} m_n\tau_n^2 + \sum_{s\in\mathcal{S}_o} m_s\tau_s^2\right].\end{aligned}$$

In order to use common optimizers we can tetermine the gradient of $\mathbb{E}$ with respect to the position of a refined particle i as:

(7.168)
$$\nabla_i\mathbb{E} = 2m_i\left[\nabla_i\mathbb{E}_{\mathcal{N}} + \nabla_i\mathbb{E}_{\mathcal{S}}\right]m$$

(7.169)
$$\nabla_i\mathbb{E}_{\mathcal{N}} = \sum_{n\in\bar{\mathcal{N}}_o} m_n\left(\tau_i+\tau_n\right)\nabla_i W_{in}, \nabla_i\mathbb{E}_{\mathcal{S}} = \sum_{s\in\mathcal{S}_o} m_s\left(\tau_i+\tau_s\right)\nabla_i W_{is}.$$

And for the derivative of $\mathbb{E}$ with respect to the weight of a refined particle i, under the constraint $\sum_{s\in\mathcal{S}_o}\frac{1}{\lambda_s}=1$ as:

(7.170)
$$\begin{aligned}\frac{\partial\mathbb{E}}{\partial\lambda_i} &= \sum_{s\in\mathcal{S}_o}\left(2m_o\frac{\partial\mathbb{E}^s_{\mathcal{N}}}{\partial\lambda_i} + 2m_o\frac{\partial\mathbb{E}^s_{\mathcal{S}}}{\partial\lambda_i} + m_o\frac{\mathbb{M}_{is}}{\lambda_i^2}\right), \frac{\partial\mathbb{E}^s_{\mathcal{N}}}{\partial\lambda_i}\\ &= \sum_{n\in\bar{\mathcal{N}}_o} m_n\left[\tau_n\frac{\mathbb{M}_{is}}{\lambda_i^2}W_{ns} + \frac{1}{\lambda_s}\frac{\partial W_{ns}}{\partial\lambda_i}\left(\tau_n+\tau_s\right)\right], \frac{\partial\mathbb{E}^s_{\mathcal{S}}}{\partial\lambda_i}\\ &= \sum_{k\in\mathcal{S}_o} m_s\left[\tau_s\frac{\mathbb{M}_{ik}}{\lambda_i^2}W_{sk} + \frac{1}{\lambda_k}\frac{\partial W_{sk}}{\partial\lambda_i}\tau_s\right],\end{aligned}$$

with

(7.171)
$$\tau_n = m_o\sum_{s\in\mathcal{S}}\frac{1}{\lambda_s}W_{ns} - m_oW_{no}, \tau_s = \sum_{j\in\mathcal{N}_s} m_jW_{sj} + m_o\sum_{k\in\mathcal{S}_o}\frac{1}{\lambda_k}W_{sk} - \rho_o,$$

(7.172)
$$\frac{\partial W_{sj}}{\partial\lambda_i} = \frac{1}{6\lambda_i^2}\left(h_s\lambda_s\mathbb{M}_{is}\mathbf{1}_{\mathcal{S}_o}(s) + h_j\lambda_j\mathbb{M}_{ij}\mathbf{1}_{\mathcal{S}_o}(j)\right)\frac{\partial W_{sj}}{\partial h_{sj}}, \mathbb{M}_{ij} = \begin{cases}\frac{1}{n-1}, & i\neq j\\ -1, & \text{else,}\end{cases}$$

(7.173)
$$\frac{\partial W_{sj}}{\partial h_{sj}} = -\frac{1}{h_{sj}}\left(dW_{sj} + |\mathbf{x}_{sj}|W'_{sj}\right), W'(r,h) = \frac{C}{h^d}\frac{\partial\hat{W}(q)}{\partial q}.$$

Chapter 8

Conclusions

In this dissertation research was presented that focused on spatially adaptive Smoothed Particle Hydrodynamics simulations both in regards to their methodological basis, e.g., focused on changing resolution, and on their practical implementation, e.g., focused on computational challenges. As a whole, this allows for simulations that have significantly reduced instabilities to allow for more practical usage of spatial adaptivity (see Chapter 7), scale-invariance of boundary handling (see Chapter 6), reduced memory and computational requirements (see Chapter 4) and improved on-the-fly rendering (see Chapter 5). Overall, the research can be separate into research focused on computational challenges and research focused on the underlying methodological aspects for spatial adaptivity. Regarding computational challenges, the papers reprinted in Chapters 2, 4 and 5 contribute as:

- Chapter 2 served as the initial starting point of the research where the primary motivation was computational in nature, i.e., reducing memory requirements. To achieve this, a process was proposed to limit the support radius of particle to constrain the number of neighbors a particle has. As a result this process allowed for an upper bound to be placed on memory consumption, which was directly beneficial to spatially adaptive simulations as in those neighborhoods can be of significantly varying size from particle to particle.

- Chapter 4 built on Chapter 2 and proposed a GPU-based data structure using compact hashing to change the memory dependence of GPU-based SPH simulations from depending on the size of the simulation domain and particle resolution to depending on the number of particles. Furthermore, by using a self-similar space-filling curve, multiple hash maps can be constructed, which allows for significant speed ups for spatially adaptive simulations, whilst significantly reducing memory requirements. Finally, an improved process was proposed to constraining the support radius and as well as set of neighborhood algorithms to speed up uniform resolution simulations.

- Chapter 5 built on the compact hash map data structure of Chapter 4 by expanding the method to allow for anisotropic SPH simulations. Furthermore, a set of algorithms was proposed to allow for an efficient on-the-fly rendering of uniform and spatially adaptive SPH simulations by directly exploiting the underlying data structures. The rendering approach can render particles as spheres and isosurfaces with a fixed memory consumption per particle.

The papers reprinted in Chapters 3, 7 and 6 focused more heavily on the underlying fundamental and methodological challenges in spatial adaptivity as:

- Chapter 3 served as the seminal research of all further spatial adaptive research in this dissertation and introduced multiple contributions. These contributions were a combination of an improved temporal blending scheme, the introduction of a novel fluid quantity redistribution process to smoothen the resolution gradient, as well as an improved particle refinement process using not just one refinement step size, but multiple. Furthermore, the concept of a sizing function was utilized to determine a continuous desired resolution on a per particle basis to remove sharp resolution transitions. Overall this allowed for multiple orders of magnitude greater spatial adaptivity, but highlighted several short comings, e.g., regarding boundary handling.

- Chapter 7 built on Chapter 3 in part by proposing an improved temporal blending scheme and by introducing a local artificial viscosity term to dampen the impact of errors introduced during particle refinement. However, the main contribution was a novel approach to generating refinement patterns that, up to this point, relied heavily on intuition and, accordingly, were not readily replicable between different researchers. The novel generation process removes all requirements on intuition and can be applied to any kernel function and any number of particles and is both applicable a priori, to generate optimal fixed patterns, and during a simulation, to improve the a priori patterns for the specific local particle distribution. Overall, these contributions significantly reduced instabilities in the simulation and, accordingly, allow for much lower artificial viscosity in the flow.

- Chapter 6 addressed a long standing problem in spatial adaptivity regarding boundary handling as many widely utilized methods, e.g., particle-based boundary handling, demonstrate a strong dependence of the fluid behavior on the sampling approach and resolution of boundary geometries. The approach presented here resolves this through introducing a non-particle based boundary handling approach that demonstrates very similar, and stable, behavior across varying fluid resolutions. This is achieved through the combination of an analytic solution for planar boundary geometries, the assumption that locally boundary geometries are flat and a signed distance field based boundary representation. This approach also allows for two-way coupling of spatially adaptive fluids, through the utilization of a single-point contact model, and furthermore improves non-uniform simulations by more accurately treating small boundary features below fluid resolution.

The results overall demonstrate a significant improvement over prior work for spatially adaptive SPH simulations, e.g., the paper reprinted in Chapter 7 allows for adaptive ratios of one million to one, compared to prior work that was limited to approximately a hundred to one. They also demonstrate a significant improvement for uniform SPH simulations, e.g., the paper reprinted in Chapter 6 demonstrates a significant improvement with regards to handling of small boundary features below particle resolution. Overall, the presented research also reduces the need for intuition, e.g., for finding optimal refinement patterns, and improves the predictability of achieved results, e.g., due to scale-invariant boundary handling. However, the research also highlighted important avenues for future work including:

- *Global splitting operators* would address one of the fundamental limitations of current spatially adaptive methods, i.e., current state of the art methods replace particles individually without considering resolution changes of adjacent particles. By considering resolution change not on a per-particle level but in a region, resolution changes could yield resolution gradients much closer to the desired resolution gradient. However, including all this information in a traditional optimization process is challenging as it would involve a large amount interdependent parameters and accordingly utilizing alternative optimization approaches would be an avenue of future research.

- *Improved particle merging* would be especially beneficial in the fluid bulk where particles of different resolutions can mix without them individually finding appropriate partners for merging. This is in line with the first point, i.e., if lowering the resolution occurred in a region, and not on a per-particle level, then this could potentially reduce errors introduced by merging. Practically, this process is similar to finding a particle configuration that obeys certain flow quantities, e.g., velocity and density, for a constrained fluid.

- *Fluid bulk adaptivity* would help in applying spatial adaptivity to a broader set of problems outside of Computer Animation. Within Computer Animation virtually all spatially adaptive methods aim to improve the surface resolution, whereas in engineering problems the fluid bulk is often equally important. To achieve this sizing functions would have to be found that can adequately measure interest in points internal to the flow. Such a sizing function, however, is difficult to design as the resolution change has to occur before something happens. Accordingly, one potential approach in this regard is utilizing learning based approaches to predict if an area could soon become relevant.

- *Pressure solver convergence* is a significant practical problem as most pressure solvers are designed under an assumption of uniform resolution. Accordingly, convergence criteria are often based on arithmetic means, which are not adequate for varying resolutions as, especially in Computer Animation, particles of low resolution are generally those under higher stresses, which is not reflected in an arithmetic mean. While utilizing a mass-weighted sum has proven useful in the past, research into more advanced criteria might yield improved stability in a wider range of scenarios.

- *Diffusion process modelling* would significantly broaden the scope of spatial adaptivity but including diffusion and mixing into such a simulation is not straightforward. The main consideration here is that particles that are being merged on a resolution basis, may not be eligible to be mixed, i.e., the merging process would act as an unphysical source of mixing. One potential avenue in this regard is simulating intra-particle physics, i.e., allowing particles to represent multiple fluid kinds simultaneously.

Bibliography

[AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. “Versatile surface tension and adhesion for SPH fluids”. In: *ACM Transactions on Graphics (TOG)* 32.6 (2013), pp. 182–189 (Cited on pages 8, 31, 38, 59, 81, 107, 131, 165, 174).

[Ada+07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J Guibas. “Adaptively sampled particle fluids”. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. 2007, p. 48 (Cited on pages 1, 25, 49, 50, 54, 71, 89, 116, 148, 149).

[AHA12] Stefan Adami, Xiangyu Y. Hu, and Nikolaus A. Adams. “A generalized wall boundary condition for smoothed particle hydrodynamics”. In: *Journal of Computational Physics* 231.21 (2012), pp. 7057–7075 (Cited on pages 12, 116).

[AI08] Alexandr Andoni and Piotr Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. In: *Communications of the ACM* 51.1 (2008), p. 117 (Cited on pages 72, 90).

[Aki+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. “Versatile rigid-fluid coupling for incompressible SPH”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 62 (Cited on pages 12, 13, 31, 38, 61, 115, 116, 118, 150).

[Aki+13a] Gizem Akinci, Nadir Akinci, Edgar Oswald, and Matthias Teschner. “Adaptive surface reconstruction for SPH using 3-level uniform grids”. In: *WSCG*. Václav Skala-UNION Agency, 2013, pp. 195–204 (Cited on pages 89, 90).

[Aki+13b] Nadir Akinci, Jens Cornelis, Gizem Akinci, and Matthias Teschner. “Coupling elastic solids with smoothed particle hydrodynamics fluids”. In: *Computer Animation and Virtual Worlds* 24.3-4 (2013), pp. 195–203 (Cited on pages 13, 131).

[ATW13] Ryoichi Ando, Nils Thuerey, and Chris Wojtan. “Highly adaptive liquid simulations on tetrahedral meshes”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 103 (Cited on pages 48, 148).

[AW87] John Amanatides and Andrew Woo. “A fast voxel traversal algorithm for ray tracing”. In: *Eurographics*. Vol. 87. 3. 1987, pp. 3–10 (Cited on pages 103, 104).

[Ban+18a] Stefan Band, Christoph Gissler, Markus Ihmsen, Jens Cornelis, Andreas Peer, and Matthias Teschner. “Pressure boundaries for implicit incompressible SPH”. In: *ACM Transactions on Graphics (TOG)* 37.2 (2018), p. 14 (Cited on pages 14–17, 71, 116, 118, 128, 133, 139).

[Ban+18b] Stefan Band, Christoph Gissler, Andreas Peer, and Matthias Teschner. “MLS pressure boundaries for divergence-free and viscous SPH fluids”. In: *Computers & Graphics* 76 (2018), pp. 37–46 (Cited on pages 71, 116, 117, 128, 129, 150).

[Bar63] Maurice S. Bartlett. “Statistical estimation of density functions”. In: *Sankhyā: The Indian Journal of Statistics, Series A* (1963), pp. 245–254 (Cited on page 2).

[Ben+17] Jan Bender, Dan Koschier, Tassilo Kugelstadt, and Marcel Weiler. “A micropolar material model for turbulent SPH fluids”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–8 (Cited on pages 17, 165).

[Ben+18] Jan Bender, Dan Koschier, Tassilo Kugelstadt, and Marcel Weiler. “Turbulent micropolar SPH fluids with foam”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.6 (2018), pp. 2284–2295 (Cited on pages 8, 107).

[Ben+19] Jan Bender, Tassilo Kugelstadt, Marcel Weiler, and Dan Koschier. “Volume Maps: An Implicit Boundary Representation for SPH”. In: *Motion, Interaction and Games*. ACM. 2019, p. 26 (Cited on pages 115, 117, 124, 131, 134, 135, 137, 138).

[Ben+20] Jan Bender, Tassilo Kugelstadt, Marcel Weiler, and Dan Koschier. “Implicit frictional boundary handling for SPH”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.10 (2020), pp. 2982–2993 (Cited on page 16).

[Ben16] Jan Bender. *SPlisHSPlasH*. Accessed: 2021-31-08. 2016. URL: `http://www.interactive-graphics.de/index.php/downloads/106-splishsplash-library` (Cited on page 131).

[Ber+17] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gael Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. “A survey of surface reconstruction from point clouds”. In: *Computer Graphics Forum*. Vol. 36. 1. Wiley Online Library. 2017, pp. 301–329 (Cited on page 90).

[BGT17] Stefan Band, Christoph Gissler, and Matthias Teschner. “Moving least squares boundaries for SPH fluids”. In: *Proceedings of the 13th Workshop on Virtual Reality Interactions and Physical Simulations*. 2017, pp. 21–28 (Cited on pages 12, 13, 116, 140, 150).

[BK15] Jan Bender and Dan Koschier. “Divergence-free smoothed particle hydrodynamics”. In: *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2015, pp. 147–155 (Cited on pages 7–10, 16, 17, 19, 31, 32, 38, 50, 58, 71, 81, 89, 106, 116, 123, 128, 129, 131, 150, 153, 165).

[BKS71] Liliana I. Boneva, David Kendall, and Ivan Stefanov. “Spline transformations: Three new diagnostic aids for the statistical data-analyst”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 33.1 (1971), pp. 1–71 (Cited on page 2).

[Bli82] James F. Blinn. “A generalization of algebraic surface drawing”. In: *ACM transactions on graphics (TOG)* 1.3 (1982), pp. 235–256 (Cited on page 90).

[BOD14] David Barcarolo Daniel A .and Le Touzé, Guillaume Oger, and Florian De Vuyst. “Adaptive particle refinement and derefinement applied to the smoothed particle hydrodynamics method”. In: *Journal of Computational Physics* 273 (2014), pp. 640–657 (Cited on pages 22–24).

[BSS87] Richard H. Byrd, Robert B. Schnabel, and Gerald A. Shultz. “A trust region algorithm for nonlinearly constrained optimization”. In: *SIAM Journal on Numerical Analysis* 24.5 (1987), pp. 1152–1170 (Cited on page 167).

[BT07] Markus Becker and Matthias Teschner. “Weakly compressible SPH for free surface flows”. In: *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation* (2007), pp. 209–217 (Cited on pages 50, 71, 89).

[BTN13] Agra Barecasco, Hanifa Terissa, and Christian Fredy Naa. “Simple free-surface detection in two and three-dimensional SPH solver”. In: *arXiv preprint arXiv:1309.4290* (2013) (Cited on page 22).

[BTT09] Markus Becker, Hendrik Tessendorf, and Matthias Teschner. “Direct Forcing for Lagrangian Rigid-Fluid Coupling”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.3 (2009), pp. 493–503 (Cited on pages 11, 116).

[BYM05] Nathan Bell, Yizhou Yu, and Peter J. Mucha. “Particle-based simulation of granular materials”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM. 2005, pp. 77–86 (Cited on page 131).

[Byr+95] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208 (Cited on page 166).

[Chi+19] Laurent Chiron, Matthieu De Leffe, Guillaume Oger, and David Le Touzé. “Fast and accurate SPH modelling of 3D complex wall boundaries in viscous and non viscous flows”. In: *Computer Physics Communications* 234 (2019), pp. 93–111 (Cited on pages 13, 16, 115, 117, 119).

[Cor+14] Jens Cornelis, Markus Ihmsen, Andreas Peer, and Matthias Teschner. “IISPH-FLIP for incompressible fluids”. In: *Computer Graphics Forum*. Vol. 33. 2. Wiley Online Library. 2014, pp. 255–262 (Cited on pages 49, 50).

[Cou15] Erwin Coumans. “Bullet physics simulation”. In: *ACM SIGGRAPH 2015 Courses*. 2015, p. 1 (Cited on page 139).

[DA12] Walter Dehnen and Hossam Aly. “Improving convergence in smoothed particle hydrodynamics simulations without pairing instability”. In: *Monthly Notices of the Royal Astronomical Society* 425.2 (2012), pp. 1068–1082 (Cited on pages 4–6, 31–33, 42, 72, 73, 91, 92, 117, 118, 151, 156, 159).

[DC99] Mathieu Desbrun and Marie-Paule Cani. *Space-Time Adaptive Simulation of Highly Deformable Substances*. Research Report RR-3829. INRIA, 1999. URL: `https://hal.inria.fr/inria-00072829` (Cited on pages 50, 71, 89).

[DCG11] Jose M. Domínguez, Alejandro J.C. Crespo, and Moncho Gómez-Gesteira. “Optimization strategies for parallel CPU and GPU implementations of a meshfree particle method”. In: *arXiv preprint arXiv:1110.3711* (2011) (Cited on pages 71, 90).

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. “Volume rendering”. In: *ACM Siggraph Computer Graphics* 22.4 (1988), pp. 65–74 (Cited on page 90).

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. “Smoothed particles: A new paradigm for animating highly deformable bodies”. In: *Computer Animation and Simulation’96*. Springer, 1996, pp. 61–76 (Cited on page 150).

[Die+15] Steven Diehl, Gabriel Rockefeller, Christopher L. Fryer, David Riethmiller, and Thomas S. Statler. “Generating optimal initial conditions for Smoothed Particle Hydrodynamics simulations”. In: *Publications of the Astronomical Society of Australia* 32 (2015) (Cited on pages 12, 23).

[Dom+11] Jose M. Domínguez, Alejandro J.C. Crespo, Moncho Gómez-Gesteira, and Jean-Christophe Marongiu. “Neighbour lists in smoothed particle hydrodynamics”. In: *International Journal for Numerical Methods in Fluids* 67.12 (2011), pp. 2026–2042 (Cited on page 32).

[Dom+13] Jose M. Dominguez, Alejandro J.C. Crespo, Daniel Valdez-Balderas, Benedict D. Rogers, and Moncho Gomez-Gesteira. “New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters”. In: *Computer Physics Communications* 184.8 (2013), pp. 1848–1860 (Cited on pages 70, 89).

[FAW10] Roland Fraedrich, Stefan Auer, and Rüdiger Westermann. “Efficient high-quality volume rendering of SPH data”. In: *IEEE transactions on visualization and computer graphics* 16.6 (2010), pp. 1533–1540 (Cited on page 90).

[FB07] Jonathan A. Feldman and Javier Bonet. “Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems”. In: *International Journal for Numerical Methods in Engineering* 72.3 (2007), pp. 295–324 (Cited on pages 25, 115, 117, 148, 149, 152, 153).

[Fel06] Jonathan A. Feldman. *Dynamic refinement and boundary contact forces in smoothed particle hydrodynamics with applications in fluid flow problems*. 2006 (Cited on pages 24, 152, 153).

[Fer+13] Martin Ferrand, Dominique Laurence, Benedict D Rogers, Damien Violeau, and Christophe Kassiotis. "Unified semi-analytical wall boundary conditions for inviscid, laminar or turbulent flows in the meshless SPH method". In: *International Journal for Numerical Methods in Fluids* 71.4 (2013), pp. 446–472 (Cited on pages 13, 16, 117).

[FM15] Makoto Fujisawa and Kenjiro T. Miura. "An efficient boundary handling with a modified density calculation for SPH". In: *Computer Graphics Forum*. Vol. 34. 7. 2015, pp. 155–162 (Cited on pages 14, 71, 115, 117, 119, 150).

[Fou+19] Georgios Fourtakas, Jose M. Dominguez, Renato Vacondio, and Benedict D. Rogers. "Local uniform stencil (LUST) boundary condition for arbitrary 3-D boundaries in parallel smoothed particle hydrodynamics (SPH) models". In: *Computers & Fluids* 190 (2019), pp. 346–361 (Cited on pages 12, 16).

[FS05] Tim Foley and Jeremy Sugerman. "KD-tree acceleration structures for a GPU raytracer". In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 2005, pp. 15–22 (Cited on page 90).

[Gar+11] Ismael Garcia, Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. "Coherent parallel hashing". In: *ACM Transactions on Graphics (TOG)*. Vol. 30. 6. ACM. 2011, p. 161 (Cited on pages 72, 90).

[Gau13] Carl Friedrich Gauss. "Theoria attractionis corporum sphaeroidicorum ellipticorum homogeneorum methodus nova tractate". In: *Commentationes Societatis regiae scientiarum Gottingensis recentiores* 2 (1813) (Cited on page 13).

[GEF15] Prashant Goswami, André Eliasson, and Pontus Franzén. "Implicit Incompressible SPH on the GPU". In: *12th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS 2015), Lyon*. 2015 (Cited on pages 17, 30, 32).

[Gis+17] Christoph Gissler, Stefan Band, Andreas Peer, Markus Ihmsen, and Matthias Teschner. "Generalized drag force for particle-based simulations". In: *Computers & Graphics* 69 (2017), pp. 1–11 (Cited on pages 81, 107, 131, 165).

[Gis+19] Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. "Interlinked SPH Pressure Solvers for Strong Fluid-Rigid Coupling". In: *ACM Transactions on Graphics (TOG)* 38.1 (2019), p. 5 (Cited on pages 13, 71, 89, 116, 118, 129, 131, 137, 139, 150).

[GM77] Robert A. Gingold and Joseph J. Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars". In: *Monthly Notices of the Royal Astronomical Society* 181.3 (1977), pp. 375–389 (Cited on pages 2, 31, 50, 70, 71, 88, 89, 117).

[Gon15] Pedro Gonnet. "Efficient and scalable algorithms for smoothed particle hydrodynamics on hybrid shared/distributed-memory architectures". In: *SIAM Journal on Scientific Computing* 37.1 (2015), pp. C95–C121 (Cited on page 32).

[Gos+10] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. "Interactive SPH Simulation and Rendering on the GPU". In: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '10. Madrid, Spain: Eurographics Association, 2010, pp. 55–64 (Cited on pages 32, 70, 71, 89, 90).

[Gre10] Simon Green. "Particle simulation using cuda". In: *NVIDIA whitepaper* 6 (2010), pp. 121–128 (Cited on pages 15, 32, 39, 70, 71, 73, 75, 82–84, 89, 90, 92–94).

[Gre28] George Green. *An essay on the application of mathematical analysis to the theories of electricity and magnetism*. Wezäta-Melins Aktiebolag, 1828 (Cited on page 13).

[Gun+07] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. "Realtime ray tracing on GPU with BVH-based packet traversal". In: *2007 IEEE Symposium on Interactive Ray Tracing*. IEEE. 2007, pp. 113–118 (Cited on page 90).

[HK89] Lars Hernquist and Neal Katz. "TREESPH-A unification of SPH with the hierarchical tree method". In: *The Astrophysical Journal Supplement Series* 70 (1989), pp. 419–446 (Cited on pages 32, 42, 58, 92).

[HKK07] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. "Smoothed Particle Hydrodynamics on GPUs". In: *Computer Graphics International* (2007), pp. 63–70 (Cited on pages 32, 38, 59).

[Hoe16] Rama Karl Hoetzlein. "GVDB: Raytracing sparse voxel database structures on the GPU". In: *Proceedings of High Performance Graphics*. 2016, pp. 109–117 (Cited on pages 133, 136).

[HS13] Christopher Jon Horvath and Barbara Solenthaler. *Mass Preserving Multi-Scale SPH*. Tech. rep. Emeryville, CA: Pixar, 2013 (Cited on pages 17, 22, 23, 25, 26, 49, 50, 52, 53, 89, 99, 100, 131, 149, 165, 174).

[Ihm+10] Markus Ihmsen, Nadir Akinci, Marc Gissler, and Matthias Teschner. "Boundary Handling and Adaptive Time-stepping for PCISPH". In: *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2010)*. The Eurographics Association, 2010 (Cited on page 150).

[Ihm+11] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. "A parallel SPH implementation on multi-core CPUs". In: *Computer Graphics Forum*. Vol. 30. 1. Wiley Online Library. 2011, pp. 99–112 (Cited on pages 30–32, 70, 71, 75, 89, 90, 93).

[Ihm+13] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. “Implicit incompressible SPH”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2013), pp. 426–435 (Cited on pages 7–10, 13, 14, 16, 19, 31, 32, 38, 50, 58, 65, 71, 116, 131, 150, 151, 165).

[Ihm+14] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. “SPH Fluids in Computer Graphics”. In: *Eurographics STARS* 2 (2014), pp. 21–42 (Cited on pages 15, 31, 71, 73, 90, 92, 127).

[Imo19] Yusuke Imoto. “Unique solvability and stability analysis for incompressible smoothed particle hydrodynamics method”. In: *Computational Particle Mechanics* 6.2 (2019), pp. 297–309 (Cited on page 16).

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2019-04-05]. 2001. URL: `http://www.scipy.org/` (Cited on pages 159, 165).

[Kar12] Tero Karras. *Thinking Parallel, Part III: Tree Construction on the GPU*. Accessed: 2019-06-17. 2012. URL: `https://devblogs.nvidia.com/thinking-parallel-part-iii-tree-construction-gpu/` (Cited on pages 74, 94).

[KB17] Dan Koschier and Jan Bender. “Density maps for improved SPH boundary handling”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, p. 1 (Cited on pages 14, 71, 81, 107, 115, 117, 119, 124, 129, 130, 134, 135, 141, 150, 151, 174).

[KDB16] Dan Koschier, Crispin Deul, and Jan Bender. “Hierarchical hp-adaptive signed distance fields.” In: *Symposium on Computer Animation*. 2016, pp. 189–198 (Cited on page 137).

[Kei+06] Richard Keiser, Bart Adams, Philip Dutré, Leonidas Guibas, and Mark Pauly. *Multiresolution particle-based fluids*. Technical Report 520. Department of Computer Science, ETH Zurich, 2006, p. 10 (Cited on pages 50, 71, 89).

[KGS19] Abbas Khayyer, Hitoshi Gotoh, and Yuma Shimizu. “A projection-based particle method with optimized particle shifting for multiphase flows with large density ratios and discontinuous density fields”. In: *Computers & Fluids* 179 (2019), pp. 356–371 (Cited on pages 3, 18).

[Kli+06] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O’brien. “Fluid animation with dynamic meshes”. In: *ACM SIGGRAPH 2006 Papers*. 2006, pp. 820–825 (Cited on pages 48, 148).

[Kos+19] Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. “Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids”. In: *Eurographics 2019 - Tutorials*. The Eurographics Association, 2019, pp. 1–41 (Cited on pages 1, 2, 7, 8, 89, 91, 116, 118, 150, 151).

[Kra88] Dieter Kraft. “A software package for sequential quadratic programming”. In: *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt* (1988) (Cited on page 167).

[KSN08] Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita. “GPU-based fast ray casting for a large number of metaballs”. In: *Computer Graphics Forum*. Vol. 27. 2. Wiley Online Library. 2008, pp. 351–360 (Cited on page 90).

[Kug+21] Tassilo Kugelstadt, Jan Bender, José Antonio Fernández-Fernández, Stefan Rhys Jeske, Fabian Löschner, and Andreas Longva. “Fast Corotated Elastic SPH Solids with Implicit Zero-Energy Mode Control”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 4.3 (2021), pp. 1–21 (Cited on page 16).

[KW03] Jens Kruger and Rüdiger Westermann. “Acceleration techniques for GPU-based volume rendering”. In: *IEEE Visualization, 2003. VIS 2003*. IEEE. 2003, pp. 287–292 (Cited on pages 90, 98).

[Lag61] Joseph-Louis Lagrange. “Nouvelles recherches sur la nature et la propagation du son”. In: *Ouevres* 1 (1761), pp. 151–332 (Cited on page 13).

[Lau+09] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. “Fast BVH construction on GPUs”. In: *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library. 2009, pp. 375–384 (Cited on page 90).

[LC87] William E. Lorensen and Harvey E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *ACM siggraph computer graphics* 21.4 (1987), pp. 163–169 (Cited on pages 89, 90).

[Lee+08] Eun-Sug Lee, Charles Moulinec, Rui Xu, Damien Violeau, Dominique Laurence, and Peter K. Stansby. “Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method”. In: *Journal of Computational Physics* 227.18 (2008), pp. 8417–8436 (Cited on page 16).

[Ler+14] Agnès Leroy, Damien Violeau, Martin Ferrand, and Christophe Kassiotis. “Unified semi-analytical wall boundary conditions applied to 2-D incompressible SPH”. In: *Journal of Computational Physics* 261 (2014), pp. 106–129 (Cited on pages 13, 16, 117, 118).

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. “Simulating water and smoke with an octree data structure”. In: *ACM transactions on graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 457–462 (Cited on pages 48, 148).

[LGS09] Wladimir J. van der Laan, Simon Green, and Miguel Sainz. “Screen space fluid rendering with curvature flow”. In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 2009, pp. 91–98 (Cited on pages 38, 89, 90, 109).

[LH06] Sylvain Lefebvre and Hugues Hoppe. “Perfect spatial hashing”. In: *ACM Transactions on Graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 579–588 (Cited on pages 72, 75, 90, 95).

[Liu+21] Sinuo Liu, Xiaokun Wang, Xiaojuan Ban, Yanrui Xu, Jing Zhou, Jiří Kosinka, and Alexandru C. Telea. “Turbulent Details Simulation for SPH Fluids via Vorticity Refinement”. In: 40.1 (2021), pp. 54–67 (Cited on pages 16, 17).

[Los+08] Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. “Two-way coupled SPH and particle level set fluid simulation”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.4 (2008), pp. 797–804 (Cited on pages 11, 116).

[LRS20] Steven J. Lind, Benedict D. Rogers, and Peter K. Stansby. “Review of smoothed particle hydrodynamics: towards converged Lagrangian flow modelling”. In: *Proceedings of the Royal Society A* 476.2241 (2020) (Cited on page 18).

[Luc77] Leon B. Lucy. “A numerical approach to the testing of the fission hypothesis”. In: *The astronomical journal* 82 (1977), pp. 1013–1024 (Cited on page 31).

[LWQ15] Chen Li, ChangBo Wang, and Hong Qin. “Novel adaptive SPH with geometric subdivision for brittle fracture animation of anisotropic materials”. In: *The Visual Computer* 31.6-8 (2015), pp. 937–946 (Cited on page 149).

[Mac+14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. “Unified particle physics for real-time applications”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 153 (Cited on page 116).

[Mar+10] Salvatore Marrone, Andrea Colagrossi, David Le Touzé, and Giorgio Graziani. “Fast free-surface detection and level-set function definition in SPH solvers”. In: *Journal of Computational Physics* 229.10 (2010), pp. 3652–3663 (Cited on page 22).

[Mar+13] Salvatore Marrone, Andrea Colagrossi, Matteo Antuono, G Colicchio, and Giorgio Graziani. “An accurate SPH modeling of viscous flows around bodies at low and moderate Reynolds numbers”. In: *Journal of Computational Physics* 245 (2013), pp. 456–475 (Cited on pages 16, 142).

[May+15] Arno Mayrhofer, Martin Ferrand, Christophe Kassiotis, Damien Violeau, and François-Xavier Morel. “Unified semi-analytical wall boundary conditions in SPH: analytical extension to 3-D”. In: *Numerical Algorithms* 68.1 (2015), pp. 15–34 (Cited on pages 13, 16, 117).

[McA+11] Aleka McAdams, Andrew Selle, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. *Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2011 (Cited on page 133).

[MCG03] Matthias Müller, David Charypar, and Markus H. Gross. “Particle-based fluid simulation for interactive applications.” In: *Symposium on Computer animation*. 2003, pp. 154–159 (Cited on pages 31, 50, 71, 89, 116).

[MM13] Miles Macklin and Matthias Müller. “Position based fluids”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 1. ISSN: 07300301. DOI: 10.1145/2461912.2461984 (Cited on pages 31, 50, 71, 116, 150).

[Mon02] Joseph J. Monaghan. “SPH compressible turbulence”. In: *Monthly Notices of the Royal Astronomical Society* 335.3 (2002), pp. 843–852 (Cited on pages 19, 131, 153, 164).

[Mon05] Joseph J. Monaghan. “Smoothed particle hydrodynamics”. In: *Reports on progress in physics* 68.8 (2005), p. 1703 (Cited on pages 31, 51, 59, 62, 64, 72, 81, 91, 107, 131, 151, 157, 164, 165).

[Mon92] Joseph J. Monaghan. “Smoothed particle hydrodynamics”. In: *Annual Review of Astronomy and Astrophysics* 30.1 (1992), pp. 543–574 (Cited on pages 4, 8, 32, 64, 118, 150).

[Mor66] Guy M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. Tech. rep. International Business Machine Co. Ltd., 1966 (Cited on page 94).

[MSD07] Matthias Müller, Simon Schirm, and Stephan Duthaler. “Screen space meshes”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2007, pp. 9–15 (Cited on page 90).

[Mus+13] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. “OpenVDB: an open-source data structure and toolkit for high-resolution volumes”. In: *Acm siggraph 2013 courses*. ACM. 2013, p. 19 (Cited on pages 72, 90, 133, 136).

[New87] Isaac Newton. *Philosophiae Naturalis Principia Mathematica*. 1687 (Cited on page 3).

[NJB07] Paul Navratil, Jarrett Johnson, and Volker Bromm. “Visualization of cosmological particle-based datasets”. In: *IEEE transactions on visualization and computer graphics* 13.6 (2007), pp. 1712–1718 (Cited on page 90).

[OK12] Jens Orthmann and Andreas Kolb. “Temporal blending for adaptive SPH”. In: *Computer Graphics Forum* 31.8 (2012), pp. 2436–2449 (Cited on pages 1, 25, 27, 31, 32, 37, 39, 40, 47, 49, 50, 54, 56, 71, 89, 149, 153, 156, 163).

[Ort+13] Jens Orthmann, Hendrik Hochstetter, Julian Bader, Serkan Bayraktar, and Andreas Kolb. “Consistent surface model for SPH-based fluid transport”. In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2013, pp. 95–103 (Cited on pages 22, 23, 31, 33).

[Ost28] Mikhail Ostrogradsky. “Démonstration d’un théoreme du calcul integral”. In: *Mémoires de l’Académie impériale des sciences de St. Pétersbourg. 6e série*. 1 (1828), pp. 39–53 (Cited on page 13).

[Owe+98] J. Michael Owen, Jens V. Villumsen, Paul R. Shapiro, and Hugo Martel. “Adaptive smoothed particle hydrodynamics: Methodology. II.” In: *The Astrophysical Journal Supplement Series* 116.2 (1998), p. 155 (Cited on pages 6, 91).

[Par62] Emanuel Parzen. “On estimation of a probability density function and mode”. In: *The Annals of Mathematical Statistics* 33.3 (1962), pp. 1065–1076 (Cited on page 2).

[Pee+15] Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. “An implicit viscosity formulation for SPH fluids”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–10 (Cited on page 61).

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016 (Cited on page 107).

[Pri12] Daniel J. Price. “Smoothed particle hydrodynamics and magnetohydrodynamics”. In: *Journal of Computational Physics* 231.3 (2012), pp. 759–794 (Cited on pages 3, 4, 6, 7, 18, 72, 118, 123, 146, 153).

[RDS10] Benedict D. Rogers, Robert A. Dalrymple, and Peter K. Stansby. “Simulation of caisson breakwater movement using 2-D SPH”. In: *Journal of Hydraulic Research* 48.S1 (2010), pp. 135–141 (Cited on page 130).

[Rei+17] Stefan Reinhardt, Markus Huber, Bernhard Eberhardt, and Daniel Weiskopf. “Fully asynchronous SPH simulation”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–10 (Cited on page 150).

[RXL21] Bo Ren, Ben Xu, and Chenfeng Li. “Unified particle system for multiple-fluid flow and porous material”. In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–14 (Cited on page 4).

[SG11] Barbara Solenthaler and Markus Gross. “Two-scale particle simulation”. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 81 (Cited on pages 1, 17, 18, 25, 26, 31, 49, 50, 116, 149, 153).

[SI12] László Szécsi and Dávid Illés. “Real-Time Metaball Ray Casting with Fragment Lists.” In: *Eurographics (Short Papers)*. 2012, pp. 93–96 (Cited on page 90).

[SP08] Barbara Solenthaler and Renato Pajarola. “Density contrast SPH interfaces”. In: *SCA ’08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008) (Cited on pages 4, 31, 33).

[SP09] Barbara Solenthaler and Renato Pajarola. “Predictive-corrective incompressible SPH”. In: *ACM Transactions on Graphics (TOG)* 28.3 (2009), p. 40 (Cited on pages 19, 50, 71, 116, 149).

[Sun+19] Pengnan Sun, Andrea Colagrossi, Salvatore Marrone, Matteo Antuono, and A-Man Zhang. “A consistent approach to particle shifting in the δ-Plus-SPH model”. In: *Computer Methods in Applied Mechanics and Engineering* 348 (2019), pp. 912–934 (Cited on pages 3, 18).

[Taf+17] Angelantonio Tafuni, Jose M. Domínguez, Renato Vacondio, and Alejandro J.C. Crespo. “Accurate and efficient SPH open boundary conditions for real 3-D engineering problems”. In: *12th International SPHERIC workshop*. 2017, pp. 1–12 (Cited on pages 11, 16).

[Tes+03] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H. Gross. “Optimized spatial hashing for collision detection of deformable objects.” In: *Vmv*. Vol. 3. 2003, pp. 47–54 (Cited on pages 69, 95).

[UHT17] Kiwon Um, Xiangyu Hu, and Nils Thuerey. “Perceptual evaluation of liquid simulation methods”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 143 (Cited on page 88).

[Vac+13] Renato Vacondio, Benedict D. Rogers, Peter K. Stansby, Paolo Mignosa, and Jonathan A. Feldman. “Variable resolution for SPH: a dynamic particle coalescing and splitting scheme”. In: *Computer Methods in Applied Mechanics and Engineering* 256 (2013), pp. 132–148 (Cited on pages 16, 22, 25, 50, 56, 147, 149, 151, 155).

[Vac+16] Renato Vacondio, Benedict D. Rogers, Peter K. Stansby, and Paolo Mignosa. “Variable resolution for SPH in three dimensions: Towards optimal splitting and coalescing for dynamic adaptivity”. In: *Computer Methods in Applied Mechanics and Engineering* 300 (2016), pp. 442–460 (Cited on pages 50, 54, 149, 151, 153, 166).

[Vac+21] Renato Vacondio, Corrado Altomare, Matthieu De Leffe, Xiangyu Hu, David Le Touzé, Steven Lind, Jean-Christophe Marongiu, Salvatore Marrone, Benedict D. Rogers, and Antonio Souto-Iglesias. “Grand challenges for Smoothed Particle Hydrodynamics numerical schemes”. In: *Computational Particle Mechanics* 8.3 (2021), pp. 575–588 (Cited on pages 4, 16).

[VBC08] Giacomo Viccione, Vittorio Bovolin, and Eugenio Pugliese Carratelli. “Defining and optimizing algorithms for neighbouring particle identification in SPH fluid simulations”. In: *International Journal for Numerical Methods in Fluids* 58.6 (2008), pp. 625–638 (Cited on page 32).

[Ver67] Loup Verlet. “Computer ”experiments” on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”. In: *Physical Review* 159.1 (1967), pp. 98–103. ISSN: 0031899X (Cited on page 32).

[VR17] Renato Vacondio and Benedict D. Rogers. “Consistent iterative shifting for SPH methods”. In: *Proceedings 12th International SPHERIC Workshop, Ourense, Spain, 13*. 2017, pp. 9–15 (Cited on page 18).

[VRS12] Renato Vacondio, Benedict D. Rogers, and Peter K. Stansby. “Accurate particle splitting for smoothed particle hydrodynamics in shallow water with shock capturing”. In: *International Journal for Numerical Methods in Fluids* 69.8 (2012), pp. 1377–1410 (Cited on page 149).

[WAK20] Rene Winchenbach, Rustam Akhunov, and Andreas Kolb. “Semi-Analytic Boundary Handling Below Particle Resolution for Smoothed Particle Hydrodynamics”. In: *Proceedings of ACM SIGGRAPH Asia 2020*. 173. Dec. 2020, 173:1–173:17 (Cited on pages xviii, 1, 27, 113, 150, 165).

[WHK16] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. “Constrained Neighbor Lists for SPH-based Fluid Simulations”. In: *Proceedings of the 15th ACM SIGGRAPH/Eurographics symposium on computer animation*. July 2016 (Cited on pages 1, 27, 29, 47, 57, 72, 78, 79, 83, 84, 90, 92, 137, 173).

[WHK17] Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. “Infinite Continuous Adaptivity for Incompressible SPH”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), 102:1–102:10 (Cited on pages xviii, 1, 25, 27, 47, 70, 71, 73, 77, 81, 84, 88, 89, 92, 107, 111, 115, 116, 131, 141, 148–151, 153, 163–165, 167–174).

[Win18] Rene Winchenbach. *Fast 3x3 SVDs for GPUs and CPUs*. Accessed: 2020-08-04. 2018. URL: `https://github.com/wi-re/tbtSVD` (Cited on page 133).

[Win19] Rene Winchenbach. *openMaelstrom*. Accessed: 2020-08-04. 2019. URL: `http://www.cg.informatik.uni-siegen.de/openMaelstrom` (Cited on pages 107, 131, 165).

[WK19] Rene Winchenbach and Andreas Kolb. “Multi-Level-Memory Structures for Adaptive SPH Simulations”. In: *Vision, Modeling and Visualization*. The Eurographics Association, 2019 (Cited on pages xviii, 1, 27, 69, 87, 107, 108, 111, 131, 141, 153, 165, 173).

[WK20] Rene Winchenbach and Andreas Kolb. “Multi-Level Memory Structures for Simulating and Rendering Smoothed Particle Hydrodynamics”. In: *Computer Graphics Forum* 39.6 (2020), pp. 527–541 (Cited on pages xviii, 1, 27, 87, 106).

[WK21] Rene Winchenbach and Andreas Kolb. “Optimized Refinement for Spatially Adaptive SPH”. In: *ACM Transactions on Graphics (TOG)* 40.1 (Jan. 2021) (Cited on pages xviii, 1, 27, 147).

[WRR18] Daniel Winkler, Massoud Rezavand, and Wolfgang Rauch. “Neighbour lists for smoothed particle hydrodynamics on GPUs”. In: *Computer Physics Communications* 225 (2018), pp. 140–148 (Cited on pages 70, 89).

[Wu+17] Wei Wu, Hongping Li, Tianyun Su, Haixing Liu, and Zhihan Lv. “GPU-accelerated SPH fluids surface reconstruction using two-level spatial uniform grids”. In: *The Visual Computer* 33.11 (2017), pp. 1429–1442 (Cited on pages 89, 90).

[Wu+18] Kui Wu, Nghia P. Truong, Cem Yuksel, and Rama Hoetzlein. “Fast Fluid Simulations with Sparse Volumes on the GPU”. In: 2018 (Cited on pages 72, 90).

[XG10] Hong Xiao and Zydrunas Gimbutas. “A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions”. In: *Computers & mathematics with applications* 59.2 (2010), pp. 663–676 (Cited on pages 126, 131).

[XZY17] Xiangyun Xiao, Shuai Zhang, and Xubo Yang. “Real-time high-quality surface rendering for large scale particle-based fluids”. In: *Proceedings of the 21st ACM siggraph symposium on interactive 3D graphics and games*. 2017, pp. 1–8 (Cited on pages 89, 90).

[Yan+16] Xiao Yan, Yun-Tao Jiang, Chen-Feng Li, Ralph R. Martin, and Shi-Min Hu. “Multiphase SPH simulation for interactive fluids and solids”. In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), pp. 1–11 (Cited on page 4).

[YRS09] Mehmet Yildiz, R.A. Rook, and Afzal Suleman. “SPH with the multiple boundary tangent method”. In: *International Journal for Numerical Methods in Engineering* 77.10 (2009), pp. 1416–1438 (Cited on page 12).

[YT13] Jihun Yu and Greg Turk. “Reconstructing surfaces of particle-based fluids using anisotropic kernels”. In: *ACM Transactions on Graphics (TOG)* 32.1 (2013), p. 5 (Cited on pages 7, 81, 91, 92, 105, 107, 109, 111, 165, 166, 174).

[ZB05] Yongning Zhu and Robert Bridson. “Animating sand as a fluid”. In: *ACM Transactions on Graphics (TOG)* 24.3 (2005), pp. 965–972 (Cited on pages 52, 105, 107, 109, 140, 142).

[Zwi+01] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. “Surface splatting”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 371–378 (Cited on page 90).