TU WIEN Informatics

Adaptives Sampling in Position-Based Fluids

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Lukas Geyer, BSc
Matrikelnummer 01026408

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Univ.Ass. Dipl.-Ing. Johannes Unterguggenberger, BSc

Wien, 22. März 2022

Lukas Geyer	Michael Wimmer

TU WIEN Informatics

Adaptive Sampling in Position-Based Fluids

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Lukas Geyer, BSc
Registration Number 01026408

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Univ.Ass. Dipl.-Ing. Johannes Unterguggenberger, BSc

Vienna, 22nd March, 2022

Lukas Geyer	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Lukas Geyer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. März 2022

Lukas Geyer

Kurzfassung

Position-Based Fluids (PBF) gehören zu den Lagrange-Flüssigkeitssimulationsmethoden, basieren auf Smoothed Particle Hydrodynamics (SPH) und erweitern das Position-Based Dynamics (PBD) Framework um die Möglichkeit, Flüssigkeiten zu simulieren. PBD verwendet Constraints, um Objektpositionen so anzupassen, dass physikalische Gesetze möglichst eingehalten werden. Im Fall von PBF werden Flüssigkeiten durch Partikel repräsentiert, deren Positionen mittels Constraints so kontrolliert werden, dass eine Kompression von Teilen der Flüssigkeit unterbunden wird. Während die ursprüngliche Version von PBF fest definierte Werte für Masse und Ruhedichte für alle Partikel vorsieht, beschreibt diese Diplomarbeit eine allgemeinere und vielseitigere Variante, in der Partikel variable Mengen an Flüssigkeit repräsentieren können. Das ermöglicht es, die Flüssigkeit mit regional variierendem Detailgrad zu simulieren, wodurch Einsparungen im Bereich des Speicherverbrauchs und der Berechnungsdauer erzielt werden können. Wir beschreiben eine Vorgehensweise, in welcher der gewünschte Detailgrad jeder Flüssigkeitsregion auf seiner Distanz zum Rand der Flüssigkeit basiert. Mittels Merging und Splitting werden die Partikel dynamisch an den gewünschten Detailgrad angepasst. Weiters beschreibt diese Diplomarbeit den Zusammenhang zwischen der Partikeldichte und der in PBF verwendeten Kernelgröße sowie Methoden, um die Kernelgröße entsprechend an den lokalen Detailgrad anzupassen. Die Vor- und Nachteile dieser Methoden werden aufgezeigt und unser bester Ansatz wird einer eingehenden mathematischen Analyse unterzogen, die die erwartete Partikelanzahl sowie die zu erwartende Anzahl von Nachbarpaaren für Positionen innerhalb der Flüssigkeit abschätzt. Aus dieser Analyse geht hervor, dass bei hinreichender Flüssigkeitstiefe sowohl die Anzahl der Partikel als auch die Anzahl der Nachbarpaare im Vergleich zur ursprünglichen PBF-Version deutlich reduziert werden können. Während unsere adaptive Methode in seichten Flüssigkeiten aufgrund eines Zuwachses an zu evaluierenden benachbarten Partikeln eher unvorteilhafte Eigenschaften aufweist, geht aus unseren Berechnungen hervor, dass die Anzahl der zu evaluierenden benachbarten Partikelpaare mit zunehmender Tiefe der Flüssigkeit deutlich reduziert wird, wodurch unsere vorgeschlagene Methode für diese Anwendungsfälle zu signifikant weniger Rechenaufwand führen kann.

Abstract

Position-Based Fluids (PBF) are a Lagrangian fluid-simulation method and are an implementation of Smoothed Particle Hydrodynamics integrated into the Position-Based Dynamics (PBD) framework. In PBD, constraints applied to object positions are used to enforce a variety of physical laws. In the case of PBF, the fluid is represented by particles and constraints are added that prevent fluid compression. The original PBF method defines all particles to be of equal mass and rest density. In this thesis, we propose a method for generalizing PBF to allow particles to represent varying amounts of fluid. This enables the fluid to be simulated with regionally varying levels of detail with the intent to reduce memory consumption and to increase performance. For each fluid region, we compute the targeted level of detail based on its distance to the fluid boundary, and use merging and splitting strategies to adapt the particles accordingly. We discuss the relation of the particle density to the kernel width used in PBF and provide several approaches for adapting the kernel width to fit the local level of detail. The advantages and disadvantages of each approach are evaluated and a streamlined implementation-variant is proposed which has advantageous properties for larger bodies of fluid. This streamlined solution bases the kernel width entirely on the boundary distance. Its approach is mathematically analyzed in regard to the expected number of particles and neighbor pairs for varying fluid body sizes. The mathematical analysis as well as measurements done in our test implementation show that while our method might increase the neighbor pair count for shallow fluids, it greatly reduces the number of particles and neighbor pairs if the fluid is sufficiently deep, giving the opportunity to significantly lower the computational effort in these cases.

Contents

Kurzfassung **vii**

Abstract **ix**

Contents **xi**

1 Introduction **1**

2 Related Work **5**
2.1 Position-Based Dynamics 5
2.2 Adaptive Sampling 5

3 Mathematical Background **9**
3.1 Position-Based Dynamics 9
3.2 Position-Based Fluids 17

4 Adaptive Sampling in Position-Based Fluids **27**
4.1 Variable Sampling 27
4.2 Kernel Width Propagation 29
4.3 Particle Merging 32
4.4 Particle Splitting 35
4.5 Adaptive Sampling 38
4.6 Streamlined Variant 43

5 Implementation **47**
5.1 Algorithm 47
5.2 Particle Data Structure 48
5.3 Data Type of Particle Positions 49
5.4 Neighborhood Search 49
5.5 Challenges and Remaining Problems 54

6 Results **55**
6.1 Expected Reduction of the Number of Neighbor Pairs 56
6.2 Measurements 62

7 Conclusion and Future Work **71**

Nomenclature **73**

Acronyms **77**

Bibliography **79**

CHAPTER 1

Introduction

Since the invention of computers, they often get used to replicate our world as we know it. The ability to do millions of computations within a few milliseconds makes computers a great tool for mathematicians and physicists alike to explore and evaluate theorems and physical laws that attempt to represent everyday phenomena. Also in the field of entertainment, computers quickly gained a foothold by offering new possibilities that were not achievable with traditional boardgames. Soon after computers were able to produce visual output, computer games were created that tried to replicate real-world scenarios.

With the advance of computer technology, computer games also developed to show more and more details, which not only required more graphical power for rendering, but also better suited equations and algorithms to approximate the laws of physics. Physics engines were created to facilitate the simulation of inertia, collision of solid objects, friction, cloth, sand, or even fluids.

While both the scientific use as well as the usage for entertainment require to model our world in computers, they have different priorities: When doing scientific research, the accuracy of the model is in most cases very important. In computer games, on the other hand, it is generally sufficient to be close enough to reality that the player does not notice a too large deviation from what they would expect to see. However, in this field it is very important that the simulation can perform in real time. A researcher or engineer might be content with waiting a few minutes or even hours for the result of the simulation of a car colliding with a wall, but an interactive game demands immediate feedback so that the game can continue in real time. It is not desirable that a player needs to wait for the physically correct computation of some event (like a car collision in a racing game).

Because of this difference in goals, intricate physical phenomena like smoke, clouds, and fluids are still often approximated in video games with fast but very crude methods like images or videos prepared in advance that just get inserted into the world. This entails a

few disadvantages: A designer is needed to insert, for example, triangle meshes for puddles, simple particle systems for fire, or a skybox textured with clouds. Furthermore, the interaction of these approximations with their surrounding is limited to what the designer thought of and was willing and able to implement. Simulating fluids or gases according to physical laws would simplify the design process: Puddles would form automatically at places where they are expected, and they interact with the environment (like dropped objects creating waves) without further design steps necessary.

While the processing power as well as available memory in modern computers are still not sufficient to entirely abandon the tricks that were developed in the field of computer graphics for approximating natural phenomena, there seems to be a growing trend towards this idea. For example, traditional rendering methods struggle to replicate many effects like transparency, refraction, shadows, or ambient illumination/occlusion, and have to rely on intricate workarounds to approximate them. With the development of consumer-grade graphics cards optimized for ray tracing tasks (like the RTX-series by NVIDIA [NVI18]), simulating the actual path of light rays—which is computationally more intensive, but naturally includes the previously mentioned effects—becomes a viable option. In a similar way, the NVIDIA FleX library [Mac20] offers simulation techniques that are targeted to replace traditional approximations of physics, like for example, manually animated collapsing buildings.

NVIDIA FleX uses a unified particle representation and is based on Position-Based Dynamics (PBD) [MHHR07], which allows for simple interaction between different object types. The particles are on a much bigger scale than atoms or molecules, but they still allow for implementing simulation rules that are more general and closer to reality than, for example, having to add a specific rule: "If a stone falls into the puddle, play a video of concentric waves at this location on the puddle surface". If the stone and the puddle fluid are modeled using particles, they can naturally interact with each other using basic collision rules. However, each particle needs space in memory and has to be included in computations, so there are certain limitations to the possible number of particles. The smaller the particles are, the more details can be represented with them, but the number of particles needed to represent the same object also increases.

As a countermeasure, we can attempt to utilize methods that were devised for other, similar problems: When rendering triangle meshes, the number of triangles increases the necessary computations and therefore the render time. To keep the number of triangles low, objects can be modeled at different levels of detail. Then, some heuristic has to be created to determine which areas need a higher level of detail and which areas are less important and can get away with a lower level of detail. Using this heuristic, some models are replaced with a less detailed version. Cascaded shadow mapping [Dim07] follows the same idea: A heuristic determines which areas can get by with shadows of a lower resolution, so that the size of the high-resolution shadow map can be reduced to only cover the remaining region. Following these ideas, the number of particles in fluid simulations can be reduced by introducing a heuristic that determines areas where the particle resolution can be reduced without severely impacting the quality of the result.

This thesis focuses on fluid simulation using PBD with different levels of simulation detail.

For fluid simulation methods like Smoothed Particle Hydrodynamics (SPH), such adaptive sampling methods have already been proposed, but to our knowledge, no solution for adaptive sampling in Position-Based Fluids (PBF) exists yet. The Hierarchical Position-Based Dynamics method by Müller [Mül08] makes use of different levels of detail, but its goal is to speed up the convergence of the constraint solver and not to reduce the number of particles—all particles still exist and need to be updated each time step. Köster and Krüger [KK16] also use a level-of-detail approach in PBF, but instead of locally adapting the sampling density, they locally adapt the number of solver iterations for each particle which reduces the workload but not the required storage space.

In this thesis, we propose a method that extends PBF to allow particles to represent varying amounts of fluid. By merging fluid particles during the simulation, we reduce the overall particle count with the expectation that having fewer particles to simulate leads to a reduced computation time. A major finding of this thesis is that the particle count is not the only decisive factor for the simulation runtime: The simulation of a fluid requires the evaluation of the interactions between neighboring fluid particles, which causes the *neighbor pair count* to also be a very important factor in regard to the time each simulation step takes. Evaluating our method in 2D and 3D showed that the neighbor pair count in our method depends on the depth of the fluid. For shallow fluid bodies, our method might even *increase* the neighbor pair count. In 2D, the necessary fluid depth to achieve improvements can be reached quite easily, leading to favorable results in our measurements. For our tests in 3D, we were not able to reach the required fluid depth because of limitations in our test implementation. Nevertheless, this thesis shows in an in-depth mathematical analysis of the expected neighbor pair count that for fluids of sufficient depth, our proposed method will not only reduce the particle count but also the neighbor pair count both in 2D and 3D, leading to potential improvements in the simulation runtime.

This thesis is structured in the following way: Chapter 2 gives a short introduction to the concept of PBD, as well as to PBF. We also present some previous approaches to fluid simulation in different resolutions. In Chapter 3, we describe the fundamental PBD algorithm and the required mathematical formulae in detail, followed by a description of how fluid simulation is performed within the PBD framework. Chapter 4 contains our main contribution: First, we introduce the concept of "particle size" in Section 4.1 by defining an equation for calculating a *particle radius* that will facilitate the further discussion of fluid sample density and more. We describe the required modifications for correctly simulating fluid particles with varying masses in Section 4.2. Next, we propose a way to regionally adapt the fluid resolution using particle merging (Section 4.3) and splitting (Section 4.4). Section 4.5 describes one possible heuristic for selecting a suitable particle resolution per area: the *boundary distance*. In Section 4.6, we show that using the *boundary distance* as our heuristic of choice allows major simplifications to the method described in Section 4.2. After the description of several tested approaches with an

analysis of their drawbacks in Chapter 4, Chapter 5 lists the methods we determined as the most successful in the order they are applied in our implementation. The neighborhood search is also affected by the variations in the fluid resolution, so we discuss the problems with the neighbor search algorithm that is commonly used in PBF and suggest two alternatives in Section 5.4. Chapter 6 contains a mathematical analysis of the effects of our method on the neighbor pair count in fluid bodies, as well as an analysis of the results from our test implementation. The nomenclature lists all mathematical variables used in this thesis as a quick reference.

CHAPTER 2

Related Work

2.1 Position-Based Dynamics

In 2006, Müller et al. proposed a new method for simulating the physical behavior of different materials and object types [MHHR07]. The original paper describes how to implement the simulation of cloth in this framework, but Position-Based Dynamics (PBD) can also be used for many other applications. A survey by Bender, Müller, and Macklin from 2017 [BMM17] summarizes the results of 10 years of research in the field of PBD and shows that this framework can be used to model a wide range of phenomena. In addition to the original cloth simulation, PBD can be used to simulate hair, sand, rigid objects, deformable solids, and fluids, to just mention a few of the possible applications. The simulation of fluid within the PBD framework was proposed in 2013 by Macklin and Müller [MM13] and is called Position-Based Fluids (PBF). They base their fluid simulation on Smoothed Particle Hydrodynamics (SPH), which is a method for approximating the behavior of fluids and gases developed in 1992 by Monaghan [Mon92]. The fluid is represented by particles, and the movement of the particles creates the flow of the fluid. Therefore, PBF can be classified as a Lagrangian simulation, as opposed to the Eulerian way, which models the fluid using a static grid instead of moving particles [Bri16]. Chapter 3 gives a more in-depth description of the operation steps in PBD as well as the specifics about PBF.

2.2 Adaptive Sampling

With PBF, a fluid is represented by a group of particles. Each particle represents a small portion of the fluid by storing a fixed mass. Knowing the rest density of the fluid, the mass of each particle also defines its size. The sizes of these particles in turn determine the resolution of the fluid. Smaller particles provide more details per fluid region, while larger particles allow to represent a larger body of fluid with a smaller number of particles,

which reduces the necessary computations and the required storage. Typically, different fluid resolutions (and hence, particle sizes) would be able to provide the required details for different fluid regions. Regions without any fluid movement might not need a high level of detail, while regions with turbulence might lose important details if the particle size is too large. For example, in video games, where the goal is to deliver a visually convincing—but not necessarily physically accurate—fluid behavior, more computational resources could be assigned to the regions close to the camera and computed in higher resolution, while regions far away or outside of the view frustum could be simplified by using larger particles.

There exist some approaches to add adaptive sampling to Lagrangian particle methods. Adams et al. [APKG07] base their simulation framework on SPH. In their approach, the sample density depends on the fluid's local feature size so that a bumpy fluid surface is sampled densely, and particles deep within the fluid volume can be larger. They have proposed algorithms for particle splitting and merging. Splitting replaces a big particle with two smaller particles that are placed symmetrically around the deleted big particle. Merging replaces two neighboring particles of the same size with one larger particle. Both of these resampling methods take measures to prevent high pressure forces that would be introduced if a new particle got placed too close to another particle.

Hong et al. [HHK08] extended the hybrid Fluid-Implicit-Particle (FLIP) method by adaptive sampling. In their proposed technique, the distance of a particle from the fluid surface is computed so that particles near the surface can be split to get a higher fluid resolution. In addition to that, the fluid's deformability is rated at a particle's position by estimating the Reynolds number [Rey83]. If this "deformability factor" is high, the resolution is increased. They proposed to partition the fluid into four layers based on the distance to the fluid surface. Particles in the two outer layers are merged and split so that the outmost layer only contains particles of minimum size, and the second layer only contains particles of standard size. In short, the size of particles in these two layers only depends on the distance from the surface. In the two inner layers, the particle size also depends on the deformability factor. Merging replaces a group of smaller particles with one larger particle at the center of gravity of the smaller particles. Splitting replaces a larger particle with two or more smaller particles of equal size, that are however placed randomly within the larger particle's neighborhood.

Zhang et al. [ZSP08] proposed an SPH simulation algorithm on the GPU that also allows adaptive sampling. In their approach, only fluid surface particles are considered for splitting, and all other particles are considered for merging. Fluid surface particles get detected by measuring a particle's distance from the center of mass in a fluid neighborhood. During splitting, one large particle gets replaced with four smaller particles arranged in a tetrahedron. Merging uses a grid of static resolution: neighboring particles that belong to the same grid cell get merged to a larger particle. The size of the grid cells imposes a lower and upper limit to the possible particle sizes in this approach. In their technique, the influence of large particles on more distant neighbors is reduced. The mass correction is based on the volume of the intersection of the neighborhood region spheres.

Solenthaler and Gross [SG11] proposed an approach for SPH-based fluid simulation where a fluid subset is selected that should contain more details. This subset is then additionally simulated with a higher resolution. In their technique, geometry-driven criteria are used for selecting the high-resolution region: fluid near obstacles, fluid inside the view frustum, and the fluid surface are proposed as candidates for the high-resolution region. The decision if a particle is part of the fluid surface is—similar to the approach by Zhang et al. [ZSP08]—performed using the neighborhood's center of mass. This property is spread further into the fluid using flood fill to get a thick surface region for the additional high-resolution simulation. Large particles entering the high-resolution region generate a cube of eight small particles in the high-resolution simulation. When a large particle leaves the high-resolution region, the corresponding small particles are deleted from the high-resolution simulation.

CHAPTER 3

Mathematical Background

3.1 Position-Based Dynamics

The PBD framework [MHHR07] can be used to simulate the physical behavior of objects like solids, ropes, cloth, fluids, and smoke. These objects are often represented using particles because of their simplicity: for many applications it is sufficient to use a set of particles that only consist of a position, a velocity vector, and a mass [MMCK14, BMM17]. In contrast to force- or impulse-based simulation models [TPBF87, BET14, BFS05, Ben07], PBD reduces the complexity of the problem by decoupling the physical constraints from the time [BMM17].

Many physical constraints can be expressed without including the time as a variable: To handle collisions and prevent objects to pass through each other, a constraint that enforces particles to keep a minimum distance can be added. To simulate cloth or ropes, a constraint that enforces a certain distance between each cloth or rope particle and its neighbors can be created, as illustrated in Figure 3.1. The simulation quality of cloth can be further improved by formulating more physical properties as constraints. Bending constraints, for example, can prevent the cloth from bending beyond a certain extent.

By defining constraints, the PBD framework is provided with rules that describe valid and invalid states of the simulated material. For cloth particles, collision constraints, distance constraints, and bending constraints are established. If in a specific state of the simulation two of such cloth particles intersect with each other, then this state gets classified as *invalid* because at least one collision constraint is violated.

3.1.1 The Basic Steps of Position-Based Dynamics

There are three basic steps in PBD: First, a time step is performed on the system where the object positions are updated according to their velocity vectors. This step is oblivious to any constraints such as physical collisions, so the resulting state of the system is

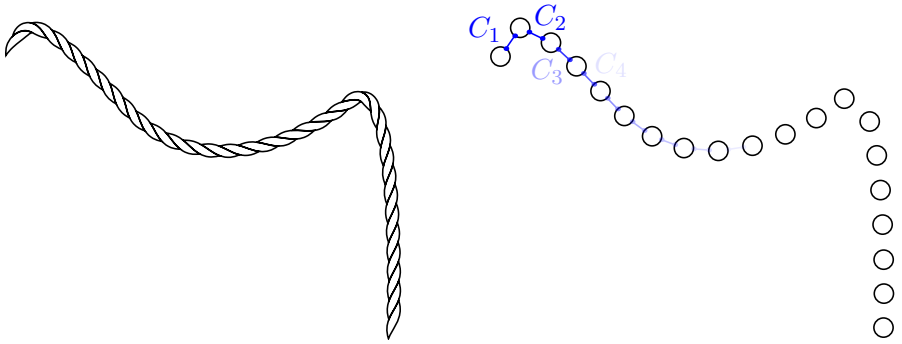
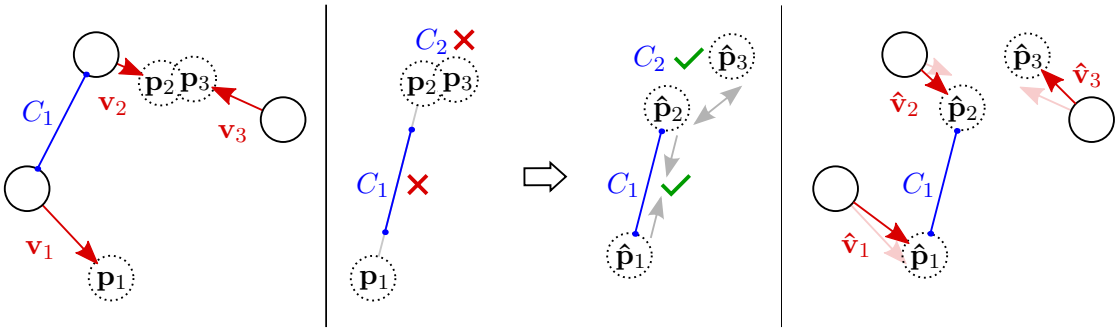


Figure 3.1: Distance constraints $C_1, C_2, \ldots$ can be used to simulate a rope. The rope is modeled using particles, and the particles are constrained to a fixed distance to their direct neighbors. Limiting the bending ability of the rope requires additional constraints.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3.2: The three steps of PBD: Step 1 predicts the new particle positions, Step 2 applies a solver to the constraints, and Step 3 updates the particle velocity vectors. The constraint C_1 is a distance constraint which gets solved by pulling $\mathbf{p}_1$ and $\mathbf{p}_2$ closer together. C_2 is a collision constraint that requires $\mathbf{p}_2$ and $\mathbf{p}_3$ to keep a certain minimum distance. It gets solved by pushing the two particles apart.

likely invalid. In the second step, only the current snapshot of the system is considered. The positions are displaced in an attempt to transfer the system into a valid state that fulfills all the constraints. As a result of the displacement, the movement trajectory of each object has changed. Therefore, the third and final step consists of updating the velocity vectors to represent the new movement direction and speed. See Figure 3.2 for an example of these three steps.

The first step which predicts the new particle position $\mathbf{p}_i$ is based only on the current position $\mathbf{x}_i$, the velocity vector $\mathbf{v}_i$, and the timestep duration Δt and can be trivially implemented as

$$\mathbf{p}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i. \tag{3.1}$$

This ensures that each particle obeys the law of inertia. The third step is similarly trivial,

where the new velocity vector $\hat{\mathbf{v}}_i$ is calculated from the final new position $\hat{\mathbf{p}}_i$ and the previous position $\mathbf{x}_i$ using

$$\hat{\mathbf{v}}_i = \frac{\hat{\mathbf{p}}_i - \mathbf{x}_i}{\Delta t}. \tag{3.2}$$

This leaves the second step to be discussed, which has to transfer a predicted particle position $\mathbf{p}_i$ to its final new position $\hat{\mathbf{p}}_i$. This step is less trivial, since it involves solving a non-linear equation system. Whether the system is non-linear depends on the constraints, but given that even the basic particle collision constraints are non-linear, it must generally be expected that the problem is non-linear. The following section goes into more detail about the constraints and how they are solved in this second step.

3.1.2 Constraints

In this thesis, constraints are represented as $C_j(\mathbf{p})$. In general there are multiple constraints, so they are indexed with j. $\mathbf{p}$ is a vector containing all scalar values that are allowed to be changed in the attempt to fulfill a constraint. The constraint $C_j(\mathbf{p})$ itself is a scalar function. We use the term "equality constraint" to refer to $C_j(\mathbf{p}) = 0$, and "inequality constraint" to refer to $C_j(\mathbf{p}) \leq 0$.[1]

In PBD, the variable values in $\mathbf{p}$ are the particle positions. Depending on the dimensionality d, each particle's position $\mathbf{p}_i$ consists of multiple scalar values. For example, if $d = 3$:

$$\mathbf{p}_i = \begin{pmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{pmatrix}. \tag{3.3}$$

In a similar way, depending on the number of particles n, the vector $\mathbf{p}$ containing all variable values consists of multiple particle positions $\mathbf{p}_i$, which in turn consist of d scalar values:

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} p_{1x} \\ p_{1y} \\ p_{1z} \end{pmatrix} \\ \vdots \\ \begin{pmatrix} p_{nx} \\ p_{ny} \\ p_{nz} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} p_{1x} \\ p_{1y} \\ p_{1z} \\ \vdots \\ p_{nx} \\ p_{ny} \\ p_{nz} \end{pmatrix}. \tag{3.4}$$

The vector $\mathbf{p}$ that is the argument of the constraint function $C_j(\mathbf{p})$ is therefore $(n \cdot d)$-dimensional. However, in many—if not most—cases, the result of the constraint

[1] In other papers, the inequality is often defined as $C_j(\mathbf{p}) \geq 0$. We, instead, choose the inequality definition to be $C_j(\mathbf{p}) \leq 0$ which is more intuitive w.r.t. fluids: There is an upper limit on the fluid density, and the constraint's value increases when the density increases.

will not depend on all of these $n \cdot d$ values. As an example, we formulate constraints that prevent particles from falling through the floor: If we define the floor to be at coordinates $y = 0$ and the coordinate system is chosen so that the positive y-axis direction points upwards, the constraints should enforce that no particle is allowed to have a y-coordinate below 0. The simplest way to capture this requirement is by creating one inequality constraint for each particle:

$$\begin{aligned} C_1(\mathbf{p}) &= -p_{1y} \leq 0 \\ C_2(\mathbf{p}) &= -p_{2y} \leq 0 \\ &\vdots \\ C_n(\mathbf{p}) &= -p_{ny} \leq 0. \end{aligned} \tag{3.5}$$

In each of these constraints, only one of the $n \cdot d$ values passed to the constraint function as an argument is actually used. For this simple case, all constraints can be trivially solved by setting p_{iy} to 0 if the corresponding constraint is violated.

To show a second, slightly more complex example, we look into creating basic constraints for modeling cloth or ropes: If a rope is represented by a string of particles, an important property is that the distance between each particle and its immediate neighbors does not change (Figure 3.1). Assuming that the particles $\mathbf{p}_1, \ldots, \mathbf{p}_n$ in the rope are equally distanced with particle distance l and ordered by their index, the equality constraints can be set up as follows:

$$\begin{aligned} C_1(\mathbf{p}) &= l - |\mathbf{p}_1 - \mathbf{p}_2| = 0 \\ C_2(\mathbf{p}) &= l - |\mathbf{p}_2 - \mathbf{p}_3| = 0 \\ &\vdots \\ C_n(\mathbf{p}) &= l - |\mathbf{p}_{n-1} - \mathbf{p}_n| = 0. \end{aligned} \tag{3.6}$$

Each of these constraints depends on $2 \cdot d$ values from $\mathbf{p}$. In this case, most values appear in two different constraints, since all except for the two particles at the ends of the rope have two neighbors. Furthermore, the constraints are now non-linear, as the distance between two positions has to be computed. This leads to a non-trivial system of non-linear equations that has to be solved.

3.1.3 Constraint Solving

The original PBD framework [MHHR07] uses a non-linear Gauss-Seidel solver approach for this task: Every constraint $C_j(\mathbf{p})$ is considered independently, one after the other. If $C_j(\mathbf{p})$ is fulfilled, nothing has to be done. If it is not, the solver attempts to improve the state of the system by performing a Newton step. $C_j(\mathbf{p})$ is locally approximated at the current state $\mathbf{p}$ with the following linearization:

$$\hat{C}_j(\mathbf{p} + \Delta\mathbf{p}) = C_j(\mathbf{p}) + \nabla C_j(\mathbf{p})\Delta\mathbf{p}. \tag{3.7}$$

If every particle has the same mass, $\Delta\mathbf{p}$ is restricted to be in the direction of the steepest constraint value change, which is the gradient $\nabla C_j(\mathbf{p})$. In this manner, the particle displacement is kept as low as possible while solving this simplified model. We first describe how to solve a constraint while ignoring the particles' masses (Equations 3.8 to 3.11), and show afterwards how to incorporate the masses (Equations 3.12 to 3.15).

Assuming all particle masses are the same, all particles are weighted equally, and $\Delta\mathbf{p}$ can be restricted to the direction of the gradient using a scalar value λ_j with

$$\Delta\mathbf{p} = \lambda_j \nabla C_j(\mathbf{p}). \tag{3.8}$$

The linearized constraint $\hat{C}_j(\mathbf{p})$ from Equation 3.7 gets solved by finding a scalar value λ_j that fulfills

$$\hat{C}_j(\mathbf{p} + \lambda_j \nabla C_j(\mathbf{p})) = C_j(\mathbf{p}) + \lambda_j \left|\nabla C_j(\mathbf{p})\right|^2 = 0. \tag{3.9}$$

Rearranged to explicitly express λ_j, we get

$$\lambda_j = -\frac{C_j(\mathbf{p})}{\left|\nabla C_j(\mathbf{p})\right|^2}. \tag{3.10}$$

Using this λ_j, the current particle positions $\mathbf{p}$ can be shifted to a solution of the linearized constraint $\hat{C}_j(\mathbf{p})$:

$$\hat{\mathbf{p}} = \mathbf{p} + \lambda_j \nabla C_j(\mathbf{p}). \tag{3.11}$$

The new positions $\hat{\mathbf{p}}$ are then used as $\mathbf{p}$ when solving the next constraint.

If the particles differ in mass, however, the goal is not to keep the overall displacement low: Particles with lower mass should be displaced more easily than particles with higher mass. To incorporate this behavior, each element of the gradient is weighted by the inverse mass of the corresponding particle (compare to Equation 3.8):

$$\Delta\mathbf{p} = \lambda_j \mathbf{M}^{-1} \nabla C_j(\mathbf{p}), \tag{3.12}$$

where $\mathbf{M}^{-1}$ is the diagonal matrix of size $(n \cdot d) \times (n \cdot d)$ containing the inverse masses: $\mathbf{M}^{-1} = \mathrm{diag}(\frac{1}{m_1}, \frac{1}{m_1}, \frac{1}{m_1}, \ldots, \frac{1}{m_n}, \frac{1}{m_n}, \frac{1}{m_n})$. Figure 3.3 shows the effect of this weighting.

Defining $\mathbf{M}^{-1/2} = \mathrm{diag}(\frac{1}{\sqrt{m_1}}, \frac{1}{\sqrt{m_1}}, \frac{1}{\sqrt{m_1}}, \ldots, \frac{1}{\sqrt{m_n}}, \frac{1}{\sqrt{m_n}}, \frac{1}{\sqrt{m_n}})$ allows us to adapt Equations 3.9 to 3.11 with only small changes:

$$\hat{C}_j(\mathbf{p} + \lambda_j \mathbf{M}^{-1} \nabla C_j(\mathbf{p})) = C_j(\mathbf{p}) + \lambda_j \left|\mathbf{M}^{-1/2} \nabla C_j(\mathbf{p})\right|^2 = 0 \tag{3.13}$$

$$\lambda_j = -\frac{C_j(\mathbf{p})}{\left|\mathbf{M}^{-1/2} \nabla C_j(\mathbf{p})\right|^2} \tag{3.14}$$

$$\hat{\mathbf{p}} = \mathbf{p} + \lambda_j \mathbf{M}^{-1} \nabla C_j(\mathbf{p}). \tag{3.15}$$

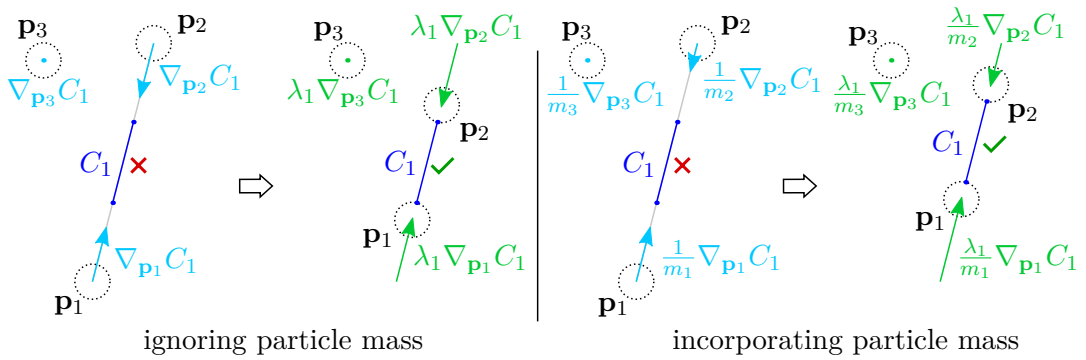


Figure 3.3: This figure shows a distance constraint C_1. When solving C_1, the displacement of each particle $\mathbf{p}_i$ is locked in the direction of $\nabla_{\mathbf{p}_i} C_1$. In this example, these directions are $\nabla_{\mathbf{p}_1} C_1$, $\nabla_{\mathbf{p}_2} C_1$ and $\nabla_{\mathbf{p}_3} C_1$. Without considering particle masses, the solver minimizes the overall displacement and shifts $\mathbf{p}_1$ and $\mathbf{p}_2$ by the same amount until the constraint is fulfilled. If the particles have different masses, for example $m_1 = 1$ and $m_2 = 2$, the gradient has to be weighted by the inverse mass, so that the solver prefers moving particles with lower mass. The right part of the illustration shows that using these weights delivers the expected result: $\mathbf{p}_2$, having double the mass of $\mathbf{p}_1$, is moved only half as far as $\mathbf{p}_1$. Particles that do not contribute to the constraint, like $\mathbf{p}_3$, have a gradient of zero and are not shifted.

Going back to the two examples from before, we will now see how to compute λ_1 in these cases. The gradient of a function C_j with multidimensional input $\mathbf{p}$ and one-dimensional output is a vector consisting of all the partial derivatives:

$$\nabla C_j(\mathbf{p}) = \begin{pmatrix} \frac{\partial C_j}{\partial p_1} \\ \vdots \\ \frac{\partial C_j}{\partial p_n} \end{pmatrix}. \tag{3.16}$$

The gradient for $C_1(\mathbf{p})$ from Equation 3.5 is therefore

$$\nabla C_1(\mathbf{p}) = \begin{pmatrix} 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.17}$$

The partial derivative is 0 for each component, except for the y-coordinate of the first particle p_{1y}, which is the second element in the $(n \cdot d)$-dimensional vector $\mathbf{p}$. The information contained in this gradient is that p_{1y} is the only parameter of influence, and that the constraint value approximately[2] increases by 1 if p_{1y} is decreased by 1. According

[2] In general, this is just an approximation, but in this case the constraints are already linear and therefore exactly match the linearized constraint approximation.

to Equation 3.14, we get $\lambda_1 = -m_1 C_1(\mathbf{p}) = m_1 p_{1y}$. Plugging this into Equation 3.15, it now states that the y-coordinate of the first particle (the second element of $\mathbf{p}$) gets increased by the constraint value according to $\nabla C_1(\mathbf{p})$, which means decreasing by p_{1y} accordingly:

$$\hat{\mathbf{p}} = \mathbf{p} + \lambda_1 \mathbf{M}^{-1} \nabla C_1(\mathbf{p}) = \mathbf{p} + p_{1y} \begin{pmatrix} 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.18}$$

This basically states to move the particle upwards by the distance it lies below the floor. As the constraint was very simple, this might seem like an excessively roundabout way to arrive at such a trivial conclusion.

For the second example—the distance constraints for simulating ropes from Equation 3.6—the gradient gets more complex:

$$C_1(\mathbf{p}) = l - |\mathbf{p}_1 - \mathbf{p}_2| = l - \sqrt{(p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2} \tag{3.19}$$

$$\nabla C_1(\mathbf{p}) = \begin{pmatrix} -\frac{p_{1x}-p_{2x}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ -\frac{p_{1y}-p_{2y}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ -\frac{p_{1z}-p_{2z}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ \frac{p_{1x}-p_{2x}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ \frac{p_{1y}-p_{2y}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ \frac{p_{1z}-p_{2z}}{\sqrt{(p_{1x}-p_{2x})^2+(p_{1y}-p_{2y})^2+(p_{1z}-p_{2z})^2}} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{p_{1x}-p_{2x}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ -\frac{p_{1y}-p_{2y}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ -\frac{p_{1z}-p_{2z}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ \frac{p_{1x}-p_{2x}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ \frac{p_{1y}-p_{2y}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ \frac{p_{1z}-p_{2z}}{|\mathbf{p}_1-\mathbf{p}_2|} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} -\left(\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|}\right) \\ \left(\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|}\right) \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{3.20}$$

The first six elements are non-zero, which means that only the first two particles have influence on the constraint value. The first three elements correspond to the position of the first particle $\mathbf{p}_1$ and form a unit vector pointing from $\mathbf{p}_1$ to $\mathbf{p}_2$. The next three elements, corresponding to $\mathbf{p}_2$, form a unit vector pointing from $\mathbf{p}_2$ to $\mathbf{p}_1$. With this, the gradient states that the constraint value will approximately[3] increase by 1 if $\mathbf{p}_1$ moves

[3]This time, the constraint is not linear. However, it *is* linear along $\nabla C_j(\mathbf{p})$, and since the shift of every individual particle is limited to this direction, the approximation is, again, exact.

towards $\mathbf{p}_2$ by a distance of 1. Similarly, it will approximately increase by 1 if $\mathbf{p}_2$ moves towards $\mathbf{p}_1$ by a distance of 1. The length of $\mathbf{M}^{-1/2}\nabla C_1(\mathbf{p})$ is $\sqrt{\frac{1}{m_1}+\frac{1}{m_2}}$, and using Equation 3.14, we get

$$\lambda_1 = -\frac{l-|\mathbf{p}_1-\mathbf{p}_2|}{\frac{1}{m_1}+\frac{1}{m_2}}. \tag{3.21}$$

Equation 3.15 then gives us

$$\hat{\mathbf{p}} = \mathbf{p} + \lambda_1\mathbf{M}^{-1}\nabla C_1(\mathbf{p}) = \mathbf{p} - \frac{l-|\mathbf{p}_1-\mathbf{p}_2|}{\frac{1}{m_1}+\frac{1}{m_2}} \begin{pmatrix} -\frac{1}{m_1}\left(\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|}\right) \\ \frac{1}{m_2}\left(\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|}\right) \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.22}$$

In general, solving the linear approximation of a constraint does not guarantee that the original constraint is also solved. Additionally, multiple constraints might depend on the same particle's position, so fulfilled constraints might become unfulfilled while solving another constraint. To mitigate this problem, the solver can iterate over all constraints multiple times with the goal of gradually converging towards a valid solution.

3.1.4 External Influence

Additionally, it is often desirable to add some external influence to the system. Examples would be gravity, which pulls every particle downwards, explicitly defined wind that pushes particles, or interaction from the user. When deciding at which time to apply these external influences, it is necessary to take into consideration what happens with the particle velocity vectors in each step of PBD. As described in Section 3.1.1, the first step applies the velocity vectors, the second step adapts the particle positions to fulfill the constraints, and the third step updates the velocity vectors based on the the particle movement that actually happened in the end. Velocity vectors are applied in step 1, and in step 3, the current velocity vector values are thrown away and replaced with ones computed from the positions. Consequently, if the external influence directly modifies the velocity vector (for example, gravity), it should be applied after step 3 and before step 1 of the subsequent timestep, or the change would get discarded. In contrast, if the external influence modifies the particle positions (for example, a teleporter), the change will not get discarded either way. However, only position changes between step 1 and 3 will influence the velocity vector. If, for example, the particle displacement should act like a teleport, and not like a movement with inertia, it has to be applied after step 3 and before step 1 of the subsequent timestep. Otherwise, it might be advisable to implement the displacement in the form of a constraint that automatically gets applied within step 2.

3.2 Position-Based Fluids

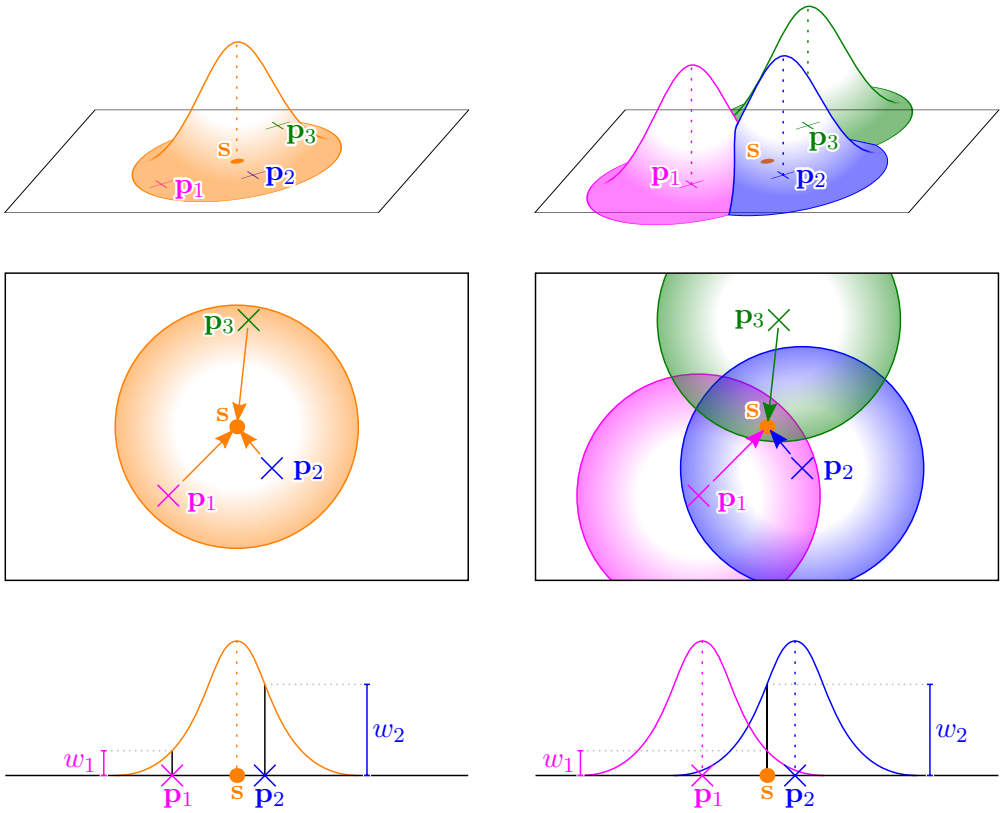
Macklin and Müller [MM13] proposed a method for simulating incompressible fluids inside the PBD framework. The constraints necessary for computing these PBF are based on SPH [Mon92]. SPH uses particles to represent the fluid, and kernel functions to calculate the fluid density at any given point.

3.2.1 Smoothed Particle Hydrodynamics

In SPH [Mon92], the fluid is represented by a finite set of particles. In the simplest case, every particle has the same mass and the particles are approximately evenly distributed within the fluid region. The particles can be seen as discrete samples of the continuous fluid. Using methods from signal processing, we can attempt to reconstruct the continuous fluid from these samples. With SPH, the use of Gaussian kernels, or alternatively approximations of Gaussian kernels, is proposed as the reconstruction filter. As a requirement, the integral of the kernel must amount to 1. Ideally, the kernel should have circular/spherical symmetry and should continually decrease from the center. To reduce the necessary computations, the spatial range of the kernel should be limited, so that only close neighbors have any influence. Common approximations of the Gaussian kernel like *poly6* and *cubic spline* [MCG03, Mon92] reach zero within finite distance. If instead the Gaussian kernel itself is used, a cutoff can be defined after which the kernel function is set to zero. We refer to the kernel radius spanning the range of non-zero values as the "kernel width".

Using the kernel, the fluid's density can be reconstructed at any given point $\mathbf{s}$, using one of two different methods (see Figure 3.4 for a visual explanation): For the first method, the kernel is centered at point $\mathbf{s}$ where the density is reconstructed, and all mass samples are weighted by this kernel and added together. In the second method, a kernel is placed at each mass sample, and the mass of the sample is distributed to its surrounding area according to the kernel. We call these two different approaches "gathering" and "spreading" of the sample masses, respectively. As long as the kernel is symmetric and of the same shape across the whole domain, both methods lead to the same result. If the integral of the kernel is 1, the resulting weighted sum of the masses is normalized to unit volume and therefore matches the density.

Not only the choice of the kernel function, also the kernel width has great influence on the resulting density, as shown in Figure 3.5. The kernel width has to be chosen depending on the density of samples: Is the kernel width too large for the sampling density, then the kernel "over-smoothes" the data and important high-frequency information is lost in the process. Is the kernel width too small, then the kernel may not contain a sufficient number of samples to return meaningful results. In the worst case, the distance between samples exceeds the kernel width, leading to disjoint droplets instead of a coherent body as the reconstructed fluid. Furthermore, as the kernel integral is fixed to be 1, decreasing the kernel width causes the kernel height to increase. As a result, the density within the disjoint droplets is over-estimated and might even surpass the fluid's rest density, as it is



(a) Kernel Gathering (b) Kernel Spreading

Figure 3.4: Two methods of applying a kernel: In (a), one kernel is centered on the sampling point $\mathbf{s}$, and the masses of the samples $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ are weighted and gathered using this kernel (kernel gathering). In (b), each of the samples $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ gets its own kernel and the mass gets spread according to its corresponding kernel to the sampling point $\mathbf{s}$ (kernel spreading). As long as all kernels have the same symmetrical shape, both methods result in the same weights w_1, w_2(, and w_3).

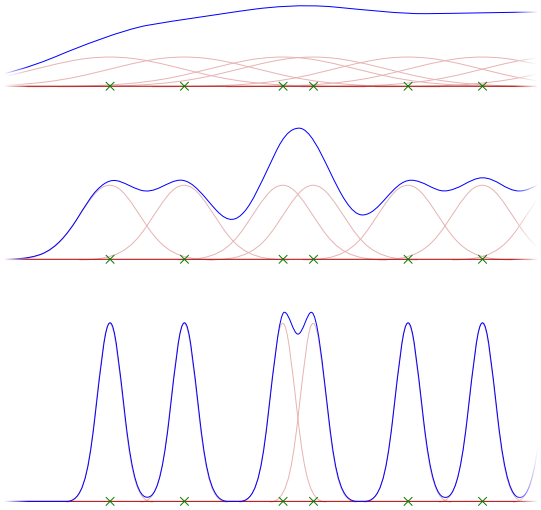


Figure 3.5: Density estimation in a one-dimensional space with three different kernel widths. The samples all have the same mass and are located at the green cross-marks. The blue line is the estimated density. The graph at the top contains large kernel widths; the estimated density is more stable, but details about the sample distribution are lost. The middle graph shows a smaller kernel width, with more density fluctuations, but also more details. In the bottom graph, the kernel width is too small to reconstruct a connected fluid, with very high density at the sample locations.

happening in the bottom row example in Figure 3.5. In Section 3.2.4, we go into more detail on how to choose a suitable kernel width.

3.2.2 Incompressibility Constraints

In general, fluids—and especially gases—are compressible, meaning that their volume can change, which allows, among other effects, sound to travel through the medium. Simulating the compressibility is, however, expensive and has a negligible effect on the macroscopic level of fluids, which the field of computer graphics is interested in. As a result, both fluids and gases are often treated as incompressible [Bri16].

Also in PBF, the goal is to enforce incompressibility on the fluid. For this purpose, incompressibility constraints are formulated that can be added to the PBD framework:

$$C_i(\mathbf{p}) = \frac{\rho_i}{\rho_0} - 1. \tag{3.23}$$

ρ_0 is the rest density of the fluid, and ρ_i is the current density. The constraints are indexed by i instead of j because we will create one constraint per particle position $\mathbf{p}_i$ for reasons that are explained in more detail below. Using i as the constraint index helps clarifying which particle it belongs to.

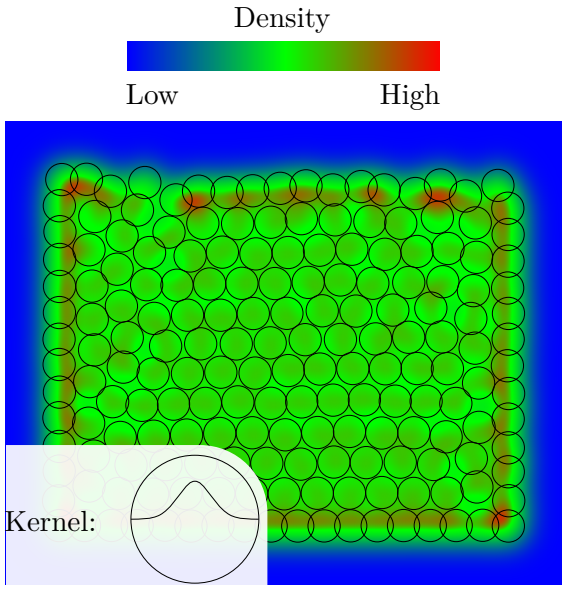


Figure 3.6: Visualization of the density within a two-dimensional fluid. In relation to the rest density, blue areas have low density, green areas have medium density, and red areas have high density. The circles indicate the particle positions. The density was computed using the Gaussian kernel plotted in the bottom left corner. The limited number of particles leads to fluctuations of the estimated density.

Each fluid particle can potentially influence the density at sampling point $\mathbf{s}$. In practice, only particles in the immediate neighborhood of $\mathbf{s}$ have an influence due to choosing a kernel with limited range, and many elements in vector $\mathbf{p}$ can be neglected in the computation of the constraint. From Equation 3.23 it can be noted, that the constraint evaluates to zero when the current density ρ_i matches the rest density ρ_0, and becomes positive when the current density ρ_i surpasses the rest density ρ_0. The original PBF method [MM13] tries to enforce constant density by using C_i as equality constraints, where each of these constraints has to evaluate to zero. Macklin et al. [MMCK14] instead use C_i as inequality constraints that have to evaluate to zero or below. These inequality constraints only enforce incompressibility, but allow a lower density which might occur at droplet particles that were isolated from the main fluid body.

Each constraint C_i enforces incompressibility only at a single position. In theory, every region in the fluid should be incompressible, so the constraint would have to be applied to infinitely many positions within the fluid. This is not only impossible to compute within a finite amount of time, it is also impossible to find a configuration that satisfies all equality constraints due to only having a finite number of particles. Density estimation using a kernel and a finite number of particles leads to fluctuations in the resulting density field, as depicted in Figure 3.6. Only enforcing inequality upon the constraints allows fluctuations below the rest density, but the problem of infinitely many constraints remains.

Consequently, the number of constrained positions has to be reduced: The particle positions are distributed throughout the fluid, so they are a natural choice for this task. For each particle, an incompressibility constraint is created that enforces incompressibility at the particle's position $\mathbf{p}_i$. To estimate the current density ρ_i at $\mathbf{p}_i$, a kernel W with width h is used:

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h), \tag{3.24}$$

where m_j is the mass of each fluid particle.

3.2.3 Incompressibility Constraint Solving

As described in Section 3.1.3, the constraint solver only considers constraints that are currently not fulfilled. Each incompressibility constraint C_i is responsible for adapting the local density at position $\mathbf{p}_i$ and only considers the kernel centered at $\mathbf{p}_i$. In case the incompressibility constraints are declared as inequality constraints, and $C_i(\mathbf{p})$ is positive, the particle positions are shifted in an attempt to decrease the constraint value to zero. In general (if the kernel continuously increases towards the center), moving a particle towards the center of the kernel will increase this kernel's density estimate. For all particles $\mathbf{p}_j$, except for $\mathbf{p}_i$, the gradient will therefore point towards the kernel center located at $\mathbf{p}_i$, or be zero for particles $\mathbf{p}_j$ outside the range of the kernel. The gradient for $\mathbf{p}_i$ is, however, a special case: Since the kernel is defined to be centered at $\mathbf{p}_i$, moving that particle also moves the whole kernel. Moving the kernel towards a particle $\mathbf{p}_j$ increases $\mathbf{p}_j$'s contribution to the estimated density by the same rate as moving $\mathbf{p}_j$ towards the kernel center, which is the exact opposite direction. Concluding from this observation, the gradient for $\mathbf{p}_i$ is the sum of all other gradients $\nabla_{\mathbf{p}_j} C_i$, negated. Figure 3.7 shows an example of a gradient $\nabla C_i(\mathbf{p})$.

In the original PBF paper [MM13], the same mass is assigned to each particle and therefore, the mass can be disregarded in the solver step. Leaving out the particle mass m_j from Equations 3.23 and 3.24, the gradient can be computed with Equation 3.25.

$$\nabla C_i(\mathbf{p}) = \frac{1}{\rho_0} \begin{pmatrix} \left(\nabla_{\mathbf{p}_1} W(\mathbf{p}_i - \mathbf{p}_1, h)\right) \\ \vdots \\ \left(\nabla_{\mathbf{p}_{i-1}} W(\mathbf{p}_i - \mathbf{p}_{i-1}, h)\right) \\ -\left(\sum_j \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right) \\ \left(\nabla_{\mathbf{p}_{i+1}} W(\mathbf{p}_i - \mathbf{p}_{i+1}, h)\right) \\ \vdots \\ \left(\nabla_{\mathbf{p}_n} W(\mathbf{p}_i - \mathbf{p}_n, h)\right) \end{pmatrix} \tag{3.25}$$

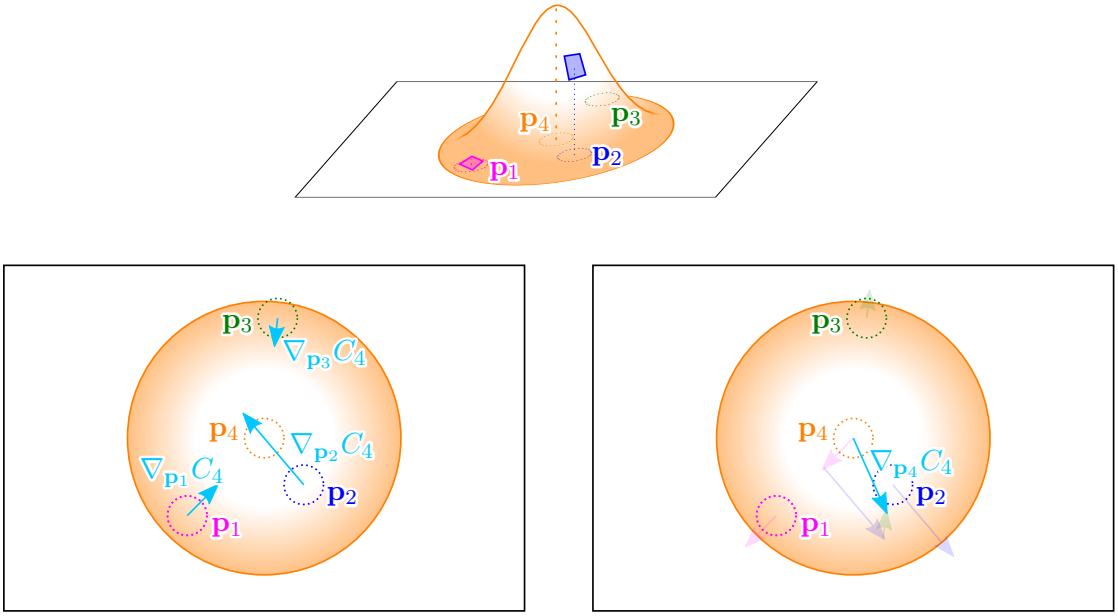


Figure 3.7: These illustrations show the gradient of an incompressibility constraint $C_4(\mathbf{p})$. The kernel is steeper at $\mathbf{p}_2$ than at $\mathbf{p}_1$ and $\mathbf{p}_3$, resulting in $\nabla_{\mathbf{p}_2} C_4$ being of higher magnitude. For $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, the constraint value increases if they are moved towards the kernel center at $\mathbf{p}_4$, as illustrated by the gradients in the bottom-left image. The bottom-right image shows the gradient for $\mathbf{p}_4$, which is the sum of the negated other three gradients. This gradient points roughly towards $\mathbf{p}_2$, because moving the kernel in this direction leads to the fastest increase of the constraint value.

According to Equation 3.10, λ_i can be computed by

$$\lambda_i = -\frac{C_i(\mathbf{p})}{|\nabla C_i(\mathbf{p})|^2} = \frac{{\rho_0}^2 - \rho_0 \sum_j W(\mathbf{p}_i - \mathbf{p}_j, h)}{\left|-\sum_j \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right|^2 + \sum_j \left|\nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right|^2}. \tag{3.26}$$

3.2.4 Kernel Width Selection

In Section 3.2.1, we mention that the kernel width has great influence on the resulting density and describe the problems caused by kernel widths that are too small or too large. If the kernel width is too small, the density constraint $C_i(\mathbf{p})$ is positive (invalid) even if no other particles than $\mathbf{p}_i$ are within the kernel, and can therefore never be fulfilled. The resulting effect is that particles entering the kernel are immediately repulsed with great force.

A kernel width that is too large results in over-smoothing, so that the exact position of the particles has a greatly reduced impact on the resulting density. If we assume two particles at the same location, and the sum of their masses weighted by the kernel height $W(\mathbf{0}, h)$ being still below the fluid's rest density, this can result in the effect that particles

pass through each other. While this can cause the particles to restlessly move around within the fluid without settling down, the main disadvantages are the reduced grade of detail (as illustrated in the top row of Figure 3.5) and the increased number of neighbor particles contained in each kernel. The large number of neighbors greatly impacts the performance of the density constraint solver.

In conclusion, the kernel width has to be chosen large enough so that the particle mass m_i weighted by the kernel height $W(\mathbf{0}, h)$ is below the rest density, but should ideally be small enough so that two particles overlapping surpass the rest density:

$$m_i W(\mathbf{0}, h) < \rho_0 < 2 m_i W(\mathbf{0}, h). \tag{3.27}$$

3.2.5 Challenges

If the incompressibility constraints are only used in the form of inequality constraints (meaning that the constraints should only push particles apart in high density regions, but not pull them closer together in low density regions), particles might get stuck on walls, depending on the formulation of the collision constraints. If the collision constraint just places every particle $\mathbf{p}_i$ lying beyond a collision plane back onto the plane, there are no neighbor particles in the incompressibility constraint C_i that can push $\mathbf{p}_i$ away from the plane. As a result, $\mathbf{p}_i$ stays stuck against the wall forever, unless the particle is moved away from the wall by some other influence (e.g., gravity or maybe some constraints that simulate surface tension).

This artifact becomes especially noticeable for vertical planar walls, where the stuck particles will stack on top of each other and stay stacked even after the rest of the fluid flowed away from the wall, as depicted in Figure 3.8. Another effect of this behavior is that in certain scenarios, the stuck particles can cause high densities: The incompressibility constraints can only move the stuck neighboring particles along the plane, but if the area along the plane in which the particles can move is also confined (for example, if the plane is the bottom of a pool which is limited in size by the pool walls), the incompressibility constraints might be unable to reduce the density. Figure 3.9 shows such a case.

A simple solution to this problem is to use a different collision constraint. One possibility is to add an additional small random offset each time a particle is repulsed from the collision plane. With this, the particles are not all on the same plane anymore and can push each other away from the wall within the collision constraints.

A second problem with the described incompressibility constraints is that whenever multiple particles simultaneously happen to match in all their properties (including position, velocity vector, and mass), the incompressibility constraints are not able to separate them again. These particles stay stuck together unless a different constraint moves them apart. The result is a high fluid density at the location of the cluster leading to undesired artifacts.

The situation of multiple particles having equal properties is only likely to occur with the aforementioned collision constraints that place particles onto a single plane. In the corner

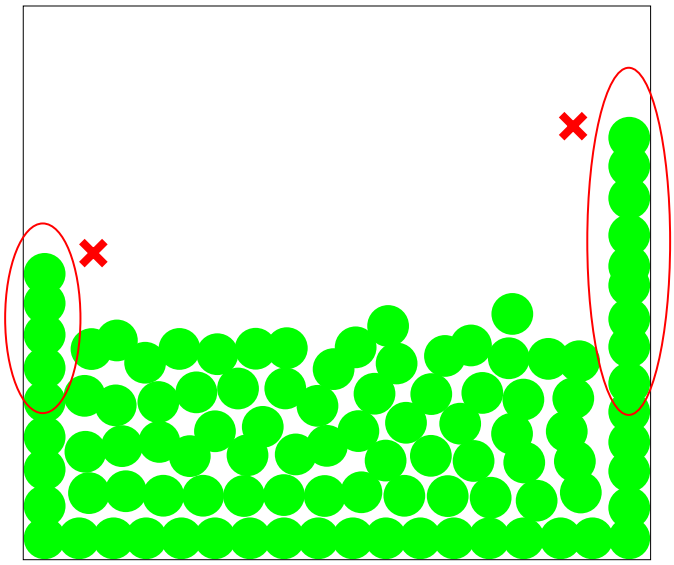


Figure 3.8: Incompressibility constraints implemented as inequality constraints are not able to push particles away from the wall. One of the resulting artifacts are particle stacks along the pool wall.

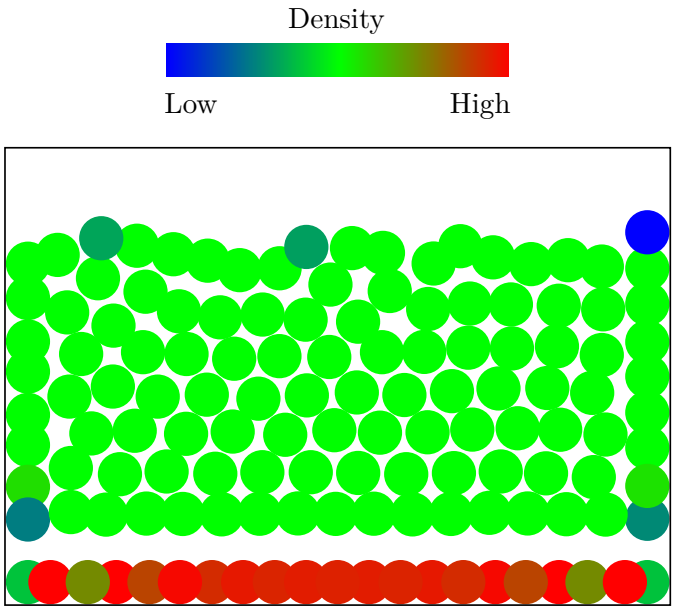


Figure 3.9: Incompressibility constraints implemented as inequality constraints are not able to push particles away from the floor. If there are more particles on the floor than allowed by the fluid's rest density, the incompressibility constraints are not able to reduce the density because they can neither push nor pull any particles away from the ground plane. Instead, all particles stick to the floor and lead to high density values in that area.

of multiple walls, several particles might be placed onto the exact same position by these collision constraints. The solution from above with the random offset also mitigates this problem.

CHAPTER 4

Adaptive Sampling in Position-Based Fluids

4.1 Variable Sampling

PBF requires that the incompressibility constraints are applied to every fluid particle in every simulation step. Fewer particles can result in a reduced amount of necessary computations and may result in increased performance. While just reducing the sampling density of fluids improves the performance, it also leads to a loss of smaller details both at the boundary of the fluid and with regard to turbulence within the fluid. One possible approach when trying to balance resolution and required workload is to introduce multiple levels of detail: In regions of high interest, the resolution is more fine-grained, while other areas can be simplified to a coarser resolution. Therefore, the sample density has to be spatially and temporally variable. In Section 3.2.1, we discussed the relation between the sample density and the kernel width (shown in Figure 3.5). If the kernel width is too small in regions with low sample density, the fluid is not correctly reconstructed as a coherent body. Using the largest required kernel width throughout the whole fluid avoids this problem, but will cause over-smoothing in areas with higher sample density. The higher precision of these areas is lost, and with it, the sole purpose of having these areas in the first place. Consequently, the kernel width also needs to vary throughout the fluid and adapt to the local sample density. This requires a way to estimate this local sample density.

To facilitate the discussion about the sample density, we assign a radius r_i to each particle. While this is an intermediary step that could in theory be skipped, it simplifies the entire concept by turning the abstract fluid samples into particle discs or balls of a certain extent. The particle radius states the expected spacing between particles and is therefore a useful measure for the local sample density. It is based on the particle's mass m_i in

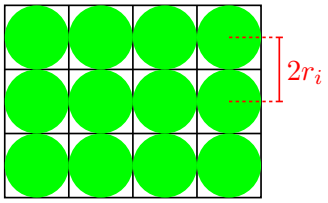


Figure 4.1: We define the particle radius r_i in a way so that for fluid particle initialization, the particles can be arranged in a regular grid with a spacing of $2r_i$ to create a fluid at rest density. Therefore, each fluid particle does not represent a disc or ball of fluid with radius r_i, but a square or cube of fluid with side length $2r_i$ (or a disc or ball with the slightly bigger radius $2\pi^{-\frac{1}{d}}r_i$ or $2(\frac{4}{3}\pi)^{-\frac{1}{d}}r_i$ with $d = 2$ for the 2D case, and $d = 3$ for the 3D case, respectively—the difference to our definition of r_i is just a constant factor). This definition of r_i is rather arbitrary, but the specific choice for the definition does not matter since we use r_i only for visualizing the particles, for re-computing the rest density, and for computing the kernel width using Equation 4.2: The rendering has no effect on the simulation, the re-computation is just the inverse of our radius definition in Equation 4.1, and Equation 4.2 scales the radius by a constant factor k which can be freely chosen to cancel out the arbitrary constant factor we introduced in the computation of the radius.

relation to its rest density ρ_{0i}:

$$r_i = \frac{1}{2}\sqrt[d]{\frac{m_i}{\rho_{0i}}}, \tag{4.1}$$

where d is the dimensionality of the simulation. Even though this thesis focuses on the simulation of one fluid type at a time (where the rest density is the same across all particles), this restriction is not a requirement for our method. Therefore, we denote the rest density not as a global constant ρ_0 as in [MM13], but as a property ρ_{0i} of each particle.

The radius r_i is chosen in a way that the equally sized particles of a fluid can be initialized by arranging them in a regular grid with a distance of $2r_i$ from each other, as depicted in Figure 4.1. In this constellation, each particle has to represent a square/cube of fluid of side length $2r_i$ (the grid cell), which is how we arrive at Equation 4.1. As a result of being defined in that way, $2r_i$ gives an approximation of the distance between fluid particles when the fluid is in rest state.

Either the rest density or the radius has to be stored as a particle property—the other value can then be computed on demand using Equation 4.1. In our implementation, we decided on storing the radius because we also use it when rendering the particles.

As discussed in Section 3.2.4, the kernel width has to be chosen adequately in relation to the particle density, and r_i represents an approximation of the expected spacing between

the particles. If every particle has the same radius, then we use a kernel width of

$$h = kr_i \tag{4.2}$$

with $k = 4$, which has led to good results in our experiments. If the radii of two particles differ, their sample density also differs and hence, their kernel widths should vary.

A first attempt might be to let the kernel width always be kr_i: If the sample density increases and the distance between the particles shrinks, the kernel width also decreases. However, this is not sufficient. Figure 4.2 shows that basing a particle's kernel width just on its radius r_i will lead to a situation where the kernels of small particles do not reach adjacent big particles. As a result, the incompressibility constraint C_1 of a small particle $\mathbf{p}_1$ will assume that the space in the direction of a big particle $\mathbf{p}_2$ is empty and let $\mathbf{p}_1$ evade pressure into this direction. The big particle $\mathbf{p}_2$, on the other hand, has a larger kernel width, which includes $\mathbf{p}_1$. Its incompressibility constraint C_2 will push back $\mathbf{p}_1$, but also cause $\mathbf{p}_2$ itself to be pushed back. In summary, C_2 works as intended, but C_1 causes wrong particle displacement due to unawareness of $\mathbf{p}_2$. The root of the problem is the kernel width being too small for the small particles that are in the neighborhood of bigger particles. As a countermeasure, we introduce *kernel width propagation*.

4.2 Kernel Width Propagation

Variable fluid sampling in PBF means that fluid particles of different sizes exist at the same time and have to correctly interact with each other. To ensure that small particles correctly interact with big particles, we propose kernel width propagation (also illustrated in Figure 4.2): As already defined in Equation 4.2, the kernel width h_i of every kernel is initialized to

$$\check{h}_i = kr_i, \tag{4.3}$$

which we call the *intrinsic kernel width* $\check{h}_i$. It serves as a lower bound for the kernel width h_i, but depending on the particle density in the vicinity of $\mathbf{p}_i$, the kernel width h_i might have to be increased. Not only does $\check{h}_i$ represent the minimal kernel width for C_i of $\mathbf{p}_i$, but also for all C_j of particles $\mathbf{p}_j$ that can contain $\mathbf{p}_i$ in their respective kernel. Therefore, $\check{h}_i$ is propagated to the neighboring particles $\mathbf{p}_j$ located within distance $\check{h}_i$ in an additional step before the constraint solver is executed. The propagated kernel widths and the $\check{h}_i$ all represent lower bounds for the kernel widths, so each particle $\mathbf{p}_i$ uses the maximum of $\check{h}_i$ and all the kernel widths it received through propagation.

Using this approach, each particle's kernel width h_i is guaranteed to be big enough for the constraint C_i to be aware of all relevant neighbor particles. However, a new problem arises: Whenever a small particle leaves the neighborhood of big particles, its kernel width instantaneously changes to a smaller value. The estimated density using this new narrow kernel might strongly deviate from the previous result using the wide kernel. Such sudden changes in the constraint value can lead to intense undesired particle movement.

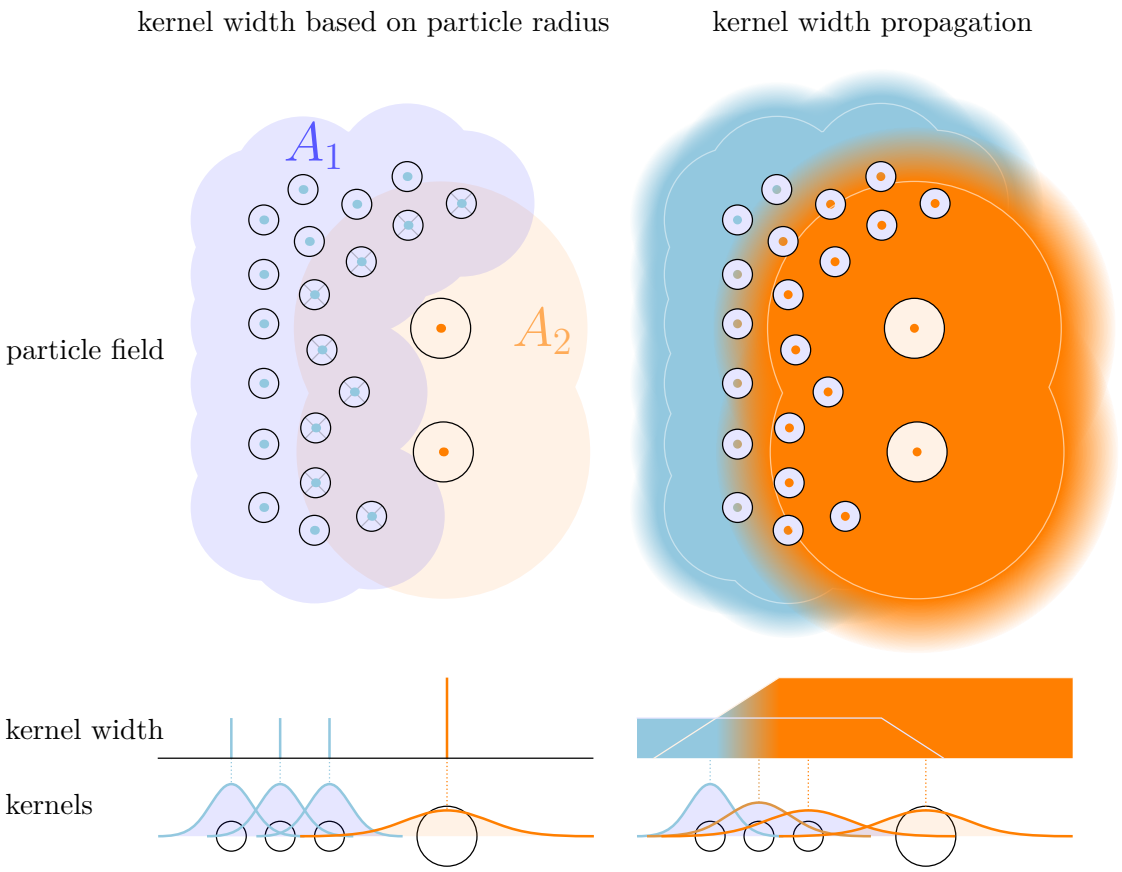


Figure 4.2: Kernel width propagation visualized for a fluid containing particles of two different sizes. Using kernel widths of kr_i with $k = 4$, the kernels of smaller particles cover area A_1, while the larger particles' kernels cover area A_2. The small particles marked with a cross are recognized as neighbors by the large kernels as they are inside A_2, but the large particles are outside of A_1 and therefore not considered in any small particle's incompressibility constraint. This leads to an underestimation of the density at the particles marked with crosses, allowing the small particles to move closer together, resulting in compression of the fluid in the region where A_1 and A_2 overlap. With *kernel width propagation*, the small particles within A_2 are assigned the same kernel width as the large particles, which means an increased kernel width compared to the situation depicted in the left image. Outside of A_2, the propagation of the large kernel width decreases linearly. Small kernel widths are also spread from A_1 outwards, but since only the maximum kernel width is used, this has no effect on the particles within A_2.

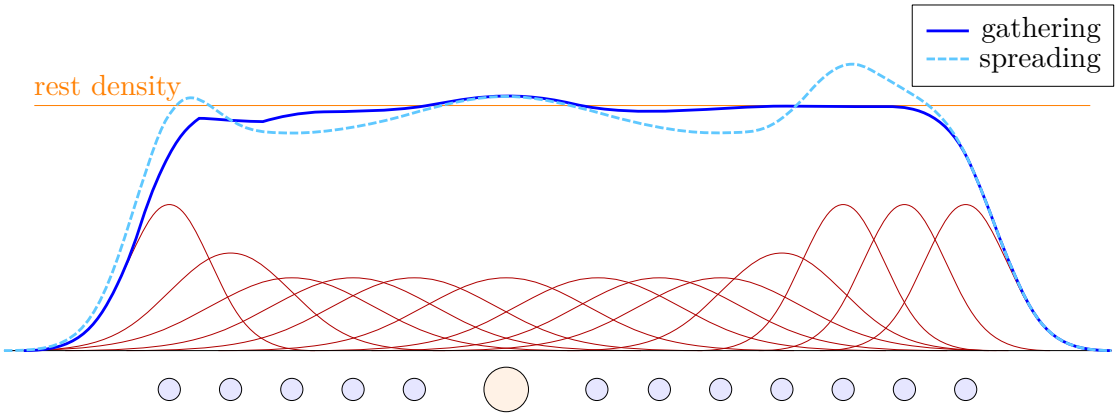


Figure 4.3: Exemplary density estimate for the fluid depicted on the bottom, consisting of 13 particles of different sizes, using the two different approaches *gathering* and *spreading*. With *gathering*, the resulting density curve matches the rest density more closely. With *spreading*, the narrow but high kernels overlap with the wide but shallow kernels at the area of transition between kernel widths, leading to an overestimation, followed by an underestimation where the narrow kernels do not reach far enough into the region of shallow kernels.

To address this, we propagate the kernel width even beyond the intrinsic kernel influence of $\check{h}_i$, but let it linearly decrease from that point on (also depicted in Figure 4.2). This creates a smooth transition between different kernel widths, and the particles have time to slowly arrange themselves to a distribution that remains valid with the reduced smoothing of a smaller kernel width. In our setup, we increase the neighborhood range by 50 % to $1.5\check{h}_i$ and let the propagated kernel width linearly decrease from $\check{h}_i$ at distance $\check{h}_i$ to 0 at distance $1.5\check{h}_i$. As before, the propagated kernel width only overwrites a neighbor's kernel width if it is the larger value.

By shifting the border of larger kernel widths further into the territory of smaller particles, every kernel is now—with respect to its surrounding particles—big enough to encompass all relevant neighboring particles for fulfilling the incompressibility constraints.

4.2.1 Spreading or Gathering

In Section 3.2.1, we discuss two different methods for the density estimation using kernels, which we call *spreading* and *gathering*. Both methods lead to the same result as long as all kernels are symmetric and of the same shape. Now that we vary the kernel width across different kernels, the shape is not the same anymore, and therefore, *spreading* and *gathering* lead to different results. Figure 4.3 shows that the results of the *gathering* method stay closer to the expected rest density, while the *spreading* method leads to stronger fluctuations. Therefore, we rely on gathering in our implementation.

4.2.2 Particle Weight

The original PBF framework [MM13] defines that all particles have the same weight and rest density, and does not incorporate the particle masses during constraint solving. In our setup, we do not restrict all fluids to have the same rest density. Furthermore, we let the particle size and weight vary locally. This requires the inclusion of the particle masses in the solver step, or otherwise light particles displace heavy particles disproportionally. When including the masses, the constraint gradient changes from Equation 3.25 to

$$\nabla C_i(\mathbf{p}) = \frac{1}{\rho_{0i}} \begin{pmatrix} m_1 \left(\nabla_{\mathbf{p}_1} W(\mathbf{p}_i - \mathbf{p}_1, h)\right) \\ \vdots \\ m_{i-1} \left(\nabla_{\mathbf{p}_{i-1}} W(\mathbf{p}_i - \mathbf{p}_{i-1}, h)\right) \\ -\left(\sum_j m_j \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right) \\ m_{i+1} \left(\nabla_{\mathbf{p}_{i+1}} W(\mathbf{p}_i - \mathbf{p}_{i+1}, h)\right) \\ \vdots \\ m_n \left(\nabla_{\mathbf{p}_n} W(\mathbf{p}_i - \mathbf{p}_n, h)\right) \end{pmatrix} \tag{4.4}$$

and using Equation 3.14, λ_i changes from Equation 3.26 to

$$\begin{aligned} \lambda_i &= -\frac{C_i(\mathbf{p})}{\left|\mathbf{M}^{-1/2} \nabla C_i(\mathbf{p})\right|^2} \\ &= \frac{{\rho_{0i}}^2 - \rho_{0i} \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h)}{\left|\frac{1}{\sqrt{m_i}} \sum_j m_j \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right|^2 + \sum_j \left|\sqrt{m_j} \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h)\right|^2}. \end{aligned} \tag{4.5}$$

These values can be used in Equation 3.15 to update the particle positions. Using these equations ensures that each constraint is solved while obeying the rule that light particles are pushed easier than heavy particles. However, each constraint is only solved once per iteration, resulting in denser regions with many small particles—and therefore many constraints—getting more importance: The dense regions cause more particle shifts due to more solved constraints C_i. As a countermeasure, we scale the particle shift in Equation 3.15 proportionally to the constraint's particle's mass m_i:

$$\hat{\mathbf{p}} = \mathbf{p} + \frac{m_i}{m_{\min}} \lambda_i \mathbf{M}^{-1} \nabla C_i(\mathbf{p}). \tag{4.6}$$

We additionally normalize this factor by the mass of the boundary particles $m_{\min}$ so that their solver equations stay unaffected and only the influence of the larger particles in the fluid center is increased.

4.3 Particle Merging

The method described in the previous section allows us to simulate a fluid with spatial variations in sampling density. The next step is to create a method for transitioning

between different sample densities. One possible approach for changing the sampling density is to increase or reduce the number of particles in certain areas, while also adapting the properties of each particle so that all particles together still represent the same amount of fluid. Particle merging and splitting can be used to achieve this. Merging and splitting generally allow to reduce or increase the particle count in an area while keeping the influence of this change local and introducing only a small amount of particle movement. Section 2.2 mentions several approaches that use different variations of particle merging and splitting. Our method implements a merging and splitting algorithm that differs from these other variations.

During particle merging, a particle s (source particle) transfers its data into another particle t (target particle) and subsequently gets deleted. The overall mass of the fluid has to stay the same, so the mass gets transferred with the formula:

$$\hat{m}_t = m_t + m_s, \tag{4.7}$$

where m_t is the mass of particle t and m_s is the mass of particle s, both before the merge, and $\hat{m}_t$ is the mass of t after the merge. As mentioned in Section 4.1, we store the radius instead of the fluid's rest density for every particle, so we also have to adapt the radius. The addition has to be performed not directly on the radius, but in regard to the volume:

$$\hat{r}_t = \sqrt[d]{{r_t}^d + {r_s}^d}. \tag{4.8}$$

r_t and r_s are the radii of particles t and s respectively, and d is the dimensionality of the simulation. $\hat{r}_t$ is the radius of t after the merge. For interpolating the particle velocity vectors, as well as the particle positions, a weighted average can be used where the weights are the particle masses:

$$\hat{\mathbf{v}}_t = \frac{m_t \mathbf{v}_t + m_s \mathbf{v}_s}{m_t + m_s} \tag{4.9}$$

$$\hat{\mathbf{p}}_t = \frac{m_t \mathbf{p}_t + m_s \mathbf{p}_s}{m_t + m_s}. \tag{4.10}$$

Sudden changes in the particle count and particle properties can lead to strong changes in the fluid density estimate. This leads to large position corrections by the incompressibility constraints and therefore results in large undesired particle velocities. One approach to mitigate this effect would be to apply the incompressibility constraints after merging happened, but to not use the resulting position changes to update the particle velocity vectors. This approach requires additional solver iterations and careful damping of particle displacement to reach a valid state. To avoid this computational overhead, we decided to merge gradually over time. In every timestep, some of the source particle's data gets transferred to the target particle, until the mass of the source particle reaches 0. At that time, the source particle is deleted.

This process can be thought of as removing a part q from the source particle and adding it to the target particle, as illustrated in Figure 4.4. q carries data like particle mass m_q,

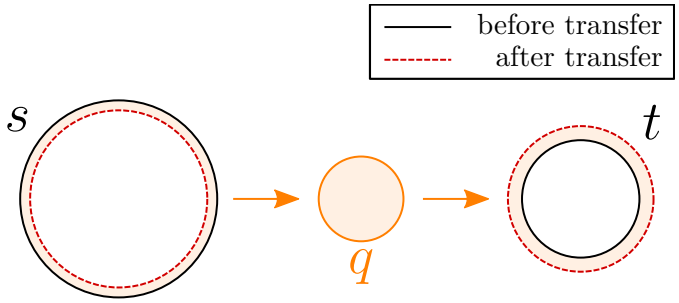


Figure 4.4: Particle transfers happen during particle merging and splitting. Each timestep of the transfer, a part q gets removed from the source particle s and added to the target particle t. The transferred data includes particle properties like the radius, position, mass, and velocity vector. The transfer's effect on the radius of s and t is shown in the illustration. The volume of the fluid has to stay constant throughout the transfer, leading to a relatively smaller radius decrease for the bigger particle s compared to the radius' increase for the smaller particle t.

radius r_q, velocity vector $\mathbf{v}_q$, and position $\mathbf{p}_q$. "Removing" this data from the source particle s has different effects depending on the property: Removing part of the mass is straightforward and can be expressed with a simple subtraction:

$$\hat{m}_s = m_s - m_q. \tag{4.11}$$

For the radius, the subtraction has to be performed after converting the radius to the corresponding volume, resulting in the equation

$$\hat{r}_s = \sqrt[d]{{r_s}^d - {r_q}^d}. \tag{4.12}$$

Removing a part q from s does not change the velocity vector of s, so $\mathbf{v}_s$ stays unchanged. Also, we do not change the position of the source particle $\mathbf{p}_s$.

Adding the data from q to the target particle t is performed in the same way as the instantaneous merging described in Equations 4.7 to 4.10, where q is the source particle that gets merged into t:

$$\hat{m}_t = m_t + m_q \tag{4.13}$$

$$\hat{r}_t = \sqrt[d]{{r_t}^d + {r_q}^d} \tag{4.14}$$

$$\hat{\mathbf{v}}_t = \frac{m_t\mathbf{v}_t + m_q\mathbf{v}_q}{m_t + m_q} \tag{4.15}$$

$$\hat{\mathbf{p}}_t = \frac{m_t\mathbf{p}_t + m_q\mathbf{p}_q}{m_t + m_q}. \tag{4.16}$$

This transfer happens each timestep, leading to a smoother transition if the transferred portion q is small enough. We decided to implement the gradual merge in a linear fashion, meaning that the transferred mass per second is constant throughout the whole merge duration. Given the remaining duration t_{left} of a merge as well as the duration Δt of the current time step, we define a factor

$$c_{\text{m}} = \min\left(\frac{\Delta t}{t_{\text{left}}}, 1\right) \tag{4.17}$$

that is used in the computation of the data of q:

$$m_q = c_{\text{m}} m_s \tag{4.18}$$

$$r_q = \sqrt[d]{c_{\text{m}} {r_s}^d} = r_s \sqrt[d]{c_{\text{m}}} \tag{4.19}$$

$$\mathbf{v}_q = \mathbf{v}_s \tag{4.20}$$

$$\mathbf{p}_q = \mathbf{p}_s. \tag{4.21}$$

The final equations for the particle property update during a gradual merge are therefore:

$$\hat{m}_t = m_t + c_{\text{m}} m_s \tag{4.22}$$

$$\hat{m}_s = m_s - c_{\text{m}} m_s = m_s(1 - c_{\text{m}}) \tag{4.23}$$

$$\hat{r}_t = \sqrt[d]{{r_t}^d + c_{\text{m}} {r_s}^d} \tag{4.24}$$

$$\hat{r}_s = \sqrt[d]{{r_s}^d - c_{\text{m}} {r_s}^d} = r_s \sqrt[d]{1 - c_{\text{m}}} \tag{4.25}$$

$$\hat{\mathbf{v}}_t = \frac{m_t \mathbf{v}_t + c_{\text{m}} m_s \mathbf{v}_s}{m_t + c_{\text{m}} m_s} \tag{4.26}$$

$$\hat{\mathbf{p}}_t = \frac{m_t \mathbf{p}_t + c_{\text{m}} m_s \mathbf{p}_s}{m_t + c_{\text{m}} m_s}. \tag{4.27}$$

Transferring the velocity and the position actually leads to worse results in our experiments, so we omit Equations 4.26 and 4.27 in our implementation. More details about the problem with position and velocity transfer are given in Section 4.4.2. We do not allow transfers to be canceled halfway through and only allow each particle to partake in at most one transfer at a time to avoid edge cases.

4.4 Particle Splitting

While particle merging allows to simplify the fluid representation in an area by reducing the sample density, there also needs to be a method for increasing the level of representable fluid detail in other areas of higher interest. Analogous to merging, we first describe how to implement splitting as a gradual process. However, in the case of splitting, this can introduce erratic particle movement, which we explain in more detail in Section 4.4.2. Section 4.4.3 describes an alternative that avoids said problem.

4.4.1 Gradual Splitting

Gradual splitting starts off by creating a new particle t of zero mass next to the particle s that gets split. This step has no impact on the local density estimate. After that, particle s transfers some of its data to particle t in every timestep until both particles have the same radius. Again using q to denote the data that gets transferred, we can re-use most of the equations in Section 4.3 just by replacing the factor c_m with a factor c_s defined as

$$c_\mathrm{s} = \min\left(\frac{\Delta t}{t_\mathrm{left}}, 1\right) \cdot \frac{{r_s}^d - {r_t}^d}{2{r_s}^d}. \tag{4.28}$$

This factor ensures a linear transfer of the mass and volume from s to t ending in both particles having the same volume (and radius). ${r_s}^d - {r_t}^d$ is the difference in volume between s and t at the current timestep; throughout t_left, half of it has to be transferred to balance their volume. The factor c_s represents a fraction of the source particle, so the transfer volume is normalized by the volume of the source particle ${r_s}^d$, leading to Equation 4.28.

In our implementation, every particle is restricted to only engage in at most one transfer (merge or split) at a time to prevent edge cases that would require special handling. For example, a particle created by a split starts off very small and would likely be chosen to immediately partake in a merge to lower the resolution. Under the added restriction, the rest densities of s and t are guaranteed to be the same.[1] As a consequence, having the same radius and volume equals to having the same mass, which allows a simplification of Equation 4.28:

$$c_\mathrm{s} = \min\left(\frac{\Delta t}{t_\mathrm{left}}, 1\right) \cdot \frac{m_s - m_t}{2m_s}. \tag{4.29}$$

All particles $\mathbf{p}_s$ ready for a split each spawn a new particle $\mathbf{p}_t$ next to them that is initialized to

$$\begin{aligned} m_t &= 0 \\ \mathbf{v}_t &= \mathbf{v}_s \end{aligned}$$

The initial positioning of the new particle requires more consideration: In Section 3.2.5, we describe that it is problematic if two particles have exactly the same position in PBF. If both particles have the same mass and position, they have the same effect on adjacent particles and will in return both receive the same displacement vectors during incompressibility constraint solving. And if both particles have the same position, mass, radius/rest density, and kernel width, each of their own incompressibility constraints will also result in the same displacement vector for themselves. As a result, these two particles will forever "stick together" unless a different constraint somehow moves them

[1] Furthermore, the merging process should ideally already be restricted to particles with the same rest density to prevent merging of different fluids.

apart. However, this problem only occurs if the position, mass, radius/rest density, and kernel width *all* match—the gradual splitting starts off both particles with different masses and radii and therefore avoids this issue.

There is still one caveat: If the two particles are far off from any adjacent particles, they will not receive different displacement vectors and the problem arises anyway. While this is unlikely to happen for many scenarios, it might still be advisable to initialize the particle position at least slightly displaced. For the displacement direction, the direction opposite to the incompressibility constraint gradient of the source particle itself can be used ($-\nabla_{\mathbf{p}_s} C_s$), as this direction points towards a region with lower density which can then be filled with the new particle from the split.

4.4.2 Problems Caused by Gradual Splitting

Unless small particles group together, they are basically playthings of the larger particles, which represent a much larger portion of fluid and therefore have a bigger influence. The large particles determine the fluid density field, and isolated small particles try to adapt their position according to this imposed density field. As a result, small particles that are located in a region of larger particles might accumulate high velocities while they attempt to find their place in the ever changing density field.

Towards the end of gradual merges and at the beginning of gradual splits, there is a short time window in which one of the two involved particles has a very small size compared to its surrounding particles. During merges, this can cause the shrinking source particle $\mathbf{p}_s$ to suddenly accumulate high velocity. The effect of this high velocity on the simulation is, however, rather low: After the merge finishes, $\mathbf{p}_s$ is removed from the simulation without further impacting the simulation with its high velocity. To also prevent any negative impact *during* the merging process, we neither transfer the velocity nor the position from $\mathbf{p}_s$ to $\mathbf{p}_t$, meaning that we omit Equations 4.26 and 4.27 in our implementation.

For splits, on the other hand, this poses a problem: The target particle starts off small and might accelerate during this phase, moving it far away from the source particle. The split continues to transfer data from $\mathbf{p}_s$ to $\mathbf{p}_t$, which equals to movement of fluid across the increasingly large distance between $\mathbf{p}_s$ and $\mathbf{p}_t$. With the growth of $\mathbf{p}_t$ while keeping its high velocity, $\mathbf{p}_t$ also gains more kinetic energy. Even if this is avoided by also transferring velocity data using Equation 4.26, the growing particle starts to push away the surrounding particles, causing undesired particle motion.

4.4.3 Instant Splitting

The core problem with gradual splitting is the time period in which one particle is small. By going back to the simpler instant splitting, this problematic time period is avoided. The source particle gets halved in volume and mass, and afterwards duplicated, resulting in two equal particles of smaller size.

There are, however, several points that need to be considered to ensure that the instant split executes smoothly: First, the split should have little impact on the density field, which can be achieved if both resulting particles are (nearly) at the same location as the source particle was, and if both particles keep the large kernel width of the source particle. Furthermore, the two resulting smaller particles must not share the same position, or else they will stick together as explained in Sections 3.2.5 and 4.4.1.

Following these requirements, the two particles are positioned close to the source particle's position, but slightly apart from each other. They can (and should) be close enough to overlap, which does not result in exceedingly high density because their kernel width is set to be the same as the source particle's (this intentionally violates Equation 3.27). To avoid a sudden change in the kernel width the next time it is updated using Equation 4.2, we let the kernel width adapt slowly to new values instead of directly assigning the result of Equation 4.2. This kernel width treatment is not limited to the particles created by a split, but is applied constantly to all fluid particles, so that abrupt kernel width changes are avoided altogether. While the kernel width slowly shrinks, the two overlapping particles start to naturally move apart without causing any sudden changes in the fluid density field.

4.5 Adaptive Sampling

We now have a method for transitioning a high-resolution area to a lower resolution and vice versa, and we are also able to let the areas of differing resolution correctly interact with each other. The remaining question is how to determine the local target resolution throughout the fluid. Previous works (see Section 2.2) often base the target resolution on the depth within the fluid, where the depth is defined as the shortest distance to the fluid surface. We consider the whole boundary of the fluid body as the surface, not only the top[2] layer of a fluid (e.g., the water surface of a lake). This includes the regions where the fluid collides with other objects, be it a pool wall, a fish, or a rock that has been dropped into the fluid and is slowly sinking to the ground. To prevent confusion, we use the term *fluid boundary* instead.

Areas deep within the fluid get a lower target resolution, whereas regions near the boundary get a high target resolution. There are two reasons for this: First, the collision of the fluid with other objects might cause increased turbulence, which requires a higher resolution for adequate representation. And second, when rendering the fluid, only the surface is visible, and a higher resolution for this area increases the visual details. To compute the depth within the fluid, a way to detect the fluid boundary is required.

4.5.1 Fluid Boundary Detection

Zhang et al. [ZSP08] classify each particle $\mathbf{p}$ as either boundary or non-boundary particle by taking all neighboring particles, computing their center of mass χ_p, and measuring

[2] assuming the fluid is under the influence of downwards gravity

the distance between χ_p and $\mathbf{p}$. If $\mathbf{p}$ is within the fluid, the neighbors are expected to be approximately evenly distributed around it, resulting in a short distance from χ_p to $\mathbf{p}$. In case $\mathbf{p}$ is located at the fluid boundary, there will be a direction without neighboring particles, and this imbalance shifts the center of mass away from $\mathbf{p}$, leading to a longer distance between χ_p and $\mathbf{p}$.

Our approach follows the same basic concept, but instead of computing the center of mass, we re-use a value that already has to be computed during incompressibility constraint solving. Equation 4.4 shows the gradient of an incompressibility constraint. Assuming the constraint applies to the kernel centered at particle $\mathbf{p}_i$, the gradient for $\mathbf{p}_i$ itself is

$$\nabla_{\mathbf{p}_i} C_i(\mathbf{p}) = -\frac{1}{\rho_{0i}} \left(\sum_j m_j \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h) \right). \tag{4.30}$$

This vector indicates the direction of greatest constraint value (density) increase for $\mathbf{p}_i$, and its length determines the rate of increase. If $\mathbf{p}_i$ is surrounded by other fluid particles, there will not be a single direction that significantly increases the density, resulting in a short vector. In case $\mathbf{p}_i$ is a boundary particle, this vector will be longer and point away from the direction where neighbor particles are "missing". In Figure 3.7, the gradient $\nabla_{\mathbf{p}_4} C_4$ is relatively long because particle $\mathbf{p}_4$ is not evenly submerged in the fluid and is therefore classified as a boundary particle. We use the length of $\nabla_{\mathbf{p}_i} C_i(\mathbf{p})$ to determine if a particle lies at the fluid boundary.

This classification has to be scale independent. For example, scaling the whole system (specifically, the positions and the particle radii) by a factor of two halves the length of the gradient. To account for this, we multiply the gradient with the kernel width before using a threshold on the gradient length to detect boundary particles.

Our gradient-based approach as well as the center-of-mass method both result in occasional misclassifications where particles within the fluid are detected as boundary particles due to the constraint solver terminating before an equilibrium is reached. These false positives are removed by checking if any of the neighbor particles $\mathbf{p}_j$ lie in the approximate opposite direction of the gradient:

$$-\frac{\nabla_{\mathbf{p}_i} C_i(\mathbf{p})}{|\nabla_{\mathbf{p}_i} C_i(\mathbf{p})|} \cdot \frac{\mathbf{p}_j - \mathbf{p}_i}{|\mathbf{p}_j - \mathbf{p}_i|} > 0.6. \tag{4.31}$$

If this condition is fulfilled by at least one neighbor particle $\mathbf{p}_j$, $\mathbf{p}_i$ is not considered a boundary particle.

One problem with the gradient-based approach (as well as with the center-of-mass approach) is that a particle without neighbors will not be classified as a boundary particle. Without neighbors, there is no change in density when moving the particle, and the gradient has length zero. Our solution addresses this by additionally considering the estimated density at the particle location. This value is also already available due to incompressibility constraint solving and is far below the rest density for isolated particles.

A similar problem occurs in areas where the fluid is only a thin film, as shown in Figure 4.5. If the fluid only consists of one layer of particles lying on a plane in a three-dimensional

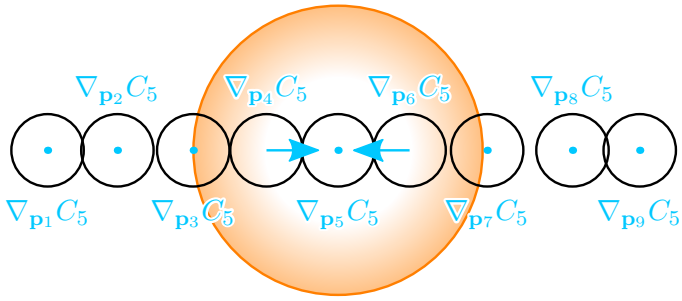


Figure 4.5: If the fluid particles form a thin film with a thickness of only one particle, the gradient $\nabla_{\mathbf{p}_i} C_i$ has a short length, which usually only happens for particles fully submerged in the fluid. In the depicted situation, the sum of all gradients $\nabla_{\mathbf{p}_j} C_5$ with $j = 1, 2, 3, 4, 6, 7, 8, 9$ results in a very short gradient $\nabla_{\mathbf{p}_5} C_5$, which would prevent a classification of $\mathbf{p}_5$ as a boundary particle. In a one-dimensional case, this classification of $\mathbf{p}_5$ is correct, but for higher dimensions, $\mathbf{p}_5$ should be a boundary particle.

environment or on a straight line in a two-dimensional environment, the gradient can be small, which would be misinterpreted as an indicator that the particle is deep within the fluid. While the gradient would normally point from the boundary deeper into the fluid, in the case of a thin film of fluid, the fluid is equally shallow in all directions. We do not handle this case explicitly, because in our test setups, these thin layers only occur in areas of low density, which already leads to a classification as boundary particles due to the additional boundary criterion introduced in the previous paragraph.

4.5.2 Boundary Distance

Knowing which particles belong to the fluid boundary now allows us to estimate the depth of each particle within the fluid. We define the depth as the shortest distance to a boundary particle and call this the *boundary distance*. By using the neighborhood relationship of the fluid particles (which has already been computed for incompressibility constraint solving), this problem turns into a multiple-source shortest path (MSSP) graph theory problem where the nodes are the particles and the edges are the neighborhood pairs with the distances as their weights. This could be solved with a fast marching algorithm, for example Dijkstra's algorithm (but modified to allow multiple sources). However, to utilize the parallel computing power of the GPU, instead of propagating the boundary distance sequentially from one particle at a time, we instead update the boundary distance by propagating the distance from *every* fluid particle to its neighbors in every iteration. Furthermore, the movement of fluid particles between each time step is small, so instead of recomputing the boundary distance from scratch every time step by performing several propagation iterations in a row, we only do one iteration per time step as an update to the old boundary distances to reduce the workload. Therefore, it takes a few time steps after the start of the simulation until the entire fluid body is populated with correct boundary distance values. At the start, every particle is initialized as a boundary particle and therefore starts with a small boundary distance. Since we

are mapping a small boundary distance to a fine level of detail, all particles start in the highest fluid resolution, which corresponds to basic PBF. During a short warmup phase, our algorithm propagates the boundary distances, evolving the simulation into our adaptively sampled variant. The duration of the warmup phase approximately equals the fluid depth divided by the kernel width of the small boundary particles, which amounts to less than 30 simulation steps (or half a second) for all examples shown in Section 6.2.

The steps in every iteration are as follows: The boundary distance of boundary particles is set to their own particle radius, and then the boundary distance of every fluid particle is propagated to its neighbors. When propagating the boundary distance b_1 of particle $\mathbf{p}_1$ to a neighbor particle $\mathbf{p}_2$, the distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ is added:

$$b_2 = b_1 + |\mathbf{p}_2 - \mathbf{p}_1| \,. \tag{4.32}$$

This boundary distance b_2 of particle $\mathbf{p}_2$ is only an approximation, because it assumes that $\mathbf{p}_1$ lies exactly in the direction of the closest boundary from $\mathbf{p}_2$. In general, a particle $\mathbf{p}_j$ will have multiple adjacent particles $\mathbf{p}_i$ that propagate their boundary distance to $\mathbf{p}_j$. Unless one of the particles $\mathbf{p}_i$ lies exactly in the direction to the closest boundary, all of the propagated boundary distances will be overestimations. To select the boundary distance closest to the truth, only the smallest propagated boundary distance is accepted.

4.5.3 Target Radius

With increasing depth into the fluid, the granularity of the fluid simulation should decrease. For this, the radius of the particles has to increase with increasing boundary distance. Using a linear function based on the boundary distance, we compute a target radius per particle. Newly created particles generally start with boundary distance b and target radius $\check{r}$ both set to r, but for particles $\mathbf{p}_t$ created during a particle split, b_t and $\check{r}_t$ are copied from the source particle $\mathbf{p}_s$.

To get all particle radii as close to their target radii as possible, the particles can be split to reduce the particle radius, and merged with a neighbor particle to increase the particle radius. We only split a particle $\mathbf{p}_s$ if after the split, the radius of both resulting smaller particles is still at least as big as the target radius $\check{r}_s$:

$$r_s \geq \sqrt[d]{2}\check{r}_s, \tag{4.33}$$

where r_s is the particle radius *before* the split.

Conversely, we only merge two adjacent particles if after the merge, the radius of the resulting bigger particle is still at most as big as the target radius $\check{r}$:

$$r_s{}^d + r_t{}^d \leq \check{r}^d, \tag{4.34}$$

where r_s and r_t are the radii of the two particles before the merge.

As we only allow each particle to partake in either one split or one merge, we do not have to take into consideration any external changes of the particle radius throughout the

merging duration. The target radius, on the other hand, might change throughout this duration, but we do not include this possibility in the merging decision making. Even if the target radius changes during the merging process, the current transfer is completed as planned. If afterwards, the target radius deviates far enough from the particle radius to fulfill one of the above conditions, the particle can partake in a new merge or split.

4.5.4 Finding Particle Pair Candidates for Merging

Equation 4.34 describes which condition the target radius has to fulfill to allow a merge of $\mathbf{p}_s$ into $\mathbf{p}_t$. Aside from this requirement, both particles should also be close to each other in order to keep the resulting particle movement minimal. Some further control over the merging pair regularity might be desired to prevent situations in which small particles will become isolated without any potential merging partner after all surrounding particles chose a different merging partner.

Our initial solution to this problem included the construction of a regular grid with different levels of detail, similar to the data structure described in Section 5.4.2. Only particles sharing the same cell on their respective level of detail would be allowed to merge. The results from following this approach exhibited suboptimal behavior, since particle pairs which would actually be ideal for being merged were often disregarded by this method because the particles were assigned to different cells. In search of a better approach, we switched to a different method that does not only work better, but is also simpler.

Instead of basing the intrinsic kernel width $\check{h}_i$ on the actual particle radius r_i as in Equation 4.3, we base it on the target radius $\check{r}_i$:

$$\check{h}_i = k\check{r}_i. \tag{4.35}$$

The effect is that particles that are smaller than their target radius—and are therefore ready for a merge—will start passing through other particles that are also ready for merging, as exemplarily shown in Figure 4.6 (the reason for particles passing through each other is explained in Section 3.2.4). With this approach, the particles arrange themselves in preparation for a potential merge. Particles that can merge together according to their target radius will start forming clusters, and particle merge pairs can be selected by finding particles overlapping each other that fulfill Equation 4.34. In our implementation, we define two particles as "overlapping" if their distance is less than any of the two particle's radii. The smaller one of the two particles is chosen as the source particle that merges into the larger target particle.

We also considered adding additional restrictions for the merging: Instead of just requiring similarity in the particles' positions, their velocity vector could also be required to match in order to avoid merging away fluid turbulence. However, we did not notice any difference in our setup, so we removed this requirement.

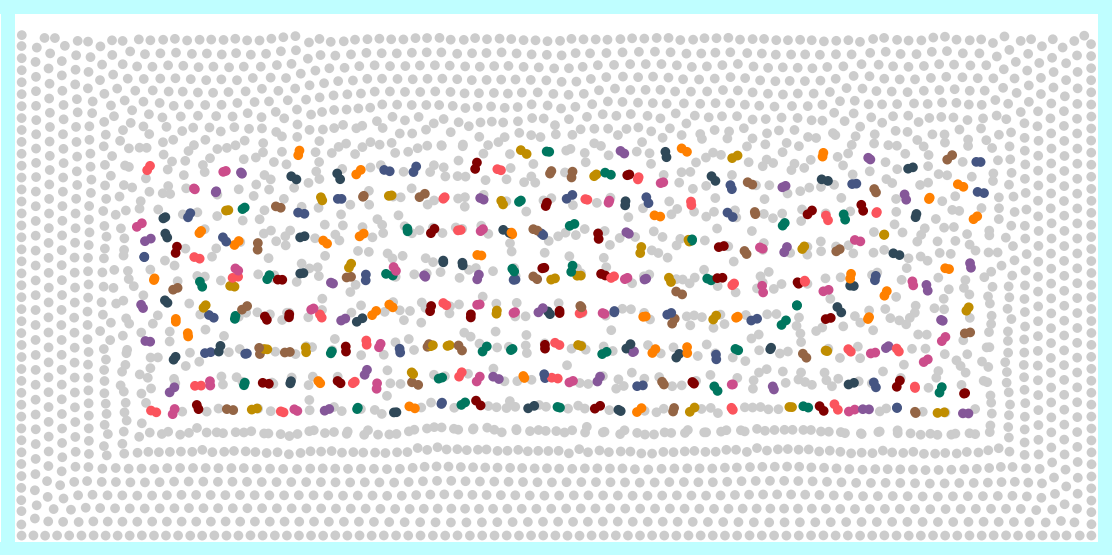


Figure 4.6: This figure shows the fluid behavior when the intrinsic kernel width is based on the target radius instead of the current radius. The target radius increases with the fluid depth, which causes the particles in the fluid center to naturally form clusters which can then be merged into bigger particles. Particle pairs currently selected for merging are highlighted in color.

4.6 Streamlined Variant

The concepts described up to now can be used to set up a fluid simulation with adaptive sampling. However, some aspects have potential for simplification thanks to the way we let the fluid's target resolution be dependent on the boundary distance. The kernel width propagation is necessary for a balanced interaction between big and small particles, but it also adds complexity as well as the need to search for neighbors in a significantly larger area than within the radius of a particle's kernel width. The target radius increases with the boundary distance, and so does the kernel width, which is in linear dependence to the target radius. But the kernel width propagation has to work into the opposite direction, to adapt the kernel width of fluid regions closer to the fluid boundary to ensure the correct interaction with deeper, larger particles.

As the coarseness of the fluid resolution (and therefore the target radius) is only dependent on the boundary distance, it is easy to predict the particle radii in the close vicinity of any given particle $\mathbf{p}_i$. This allows to predict the effects of the kernel width propagation and express the estimated final kernel width explicitly in a function only depending on the boundary distance without having to perform the propagation step. Or, looked upon from a different viewpoint, in this variant we define the kernel width to be in linear dependence to the boundary distance and find the corresponding target radii that still satisfy the conditions that were previously enforced by kernel width spreading.

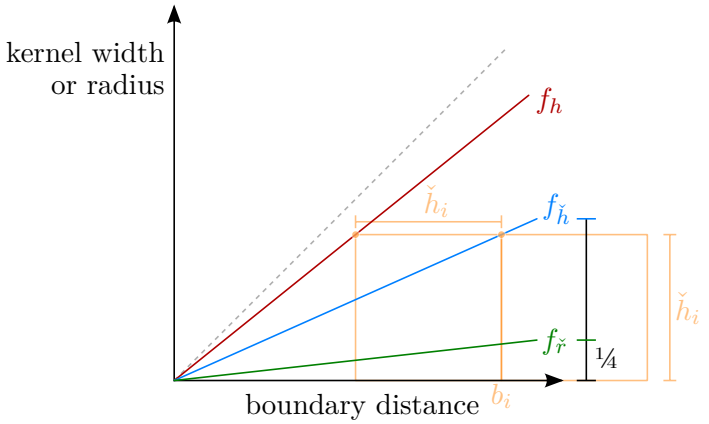


Figure 4.7: The kernel width $f_h(b)$, the intrinsic kernel width $f_{\check{h}}(b)$, and the target radius $f_{\check{r}}(b)$ can be calculated based on the boundary distance b. First, f_h is defined using Equation 4.36 with a fixed gradient a. Once f_h is decided, an $f_{\check{h}}$ has to be found that fulfills the following condition: For every particle $\mathbf{p}_i$, all of its adjacent particles $\mathbf{p}_j$ (that lie within the intrinsic kernel width $\check{h}_i = f_{\check{h}}(b_i)$) must have a kernel width $h_j = f_h(b_j)$ for which $h_j \geq \check{h}_i$ holds true, so that $\mathbf{p}_i$ is included in the neighborhood of those adjacent $\mathbf{p}_j$. The target radius $f_{\check{r}}$ is defined by Equation 4.35 (and assuming $k = 4$) as one quarter of the intrinsic kernel width $f_{\check{h}}$.

Figure 4.7 shows the kernel width plotted against the boundary distance which we define as a linear relationship f_h. Ideally, every kernel lies completely inside the fluid for an accurate density estimation. Therefore, the kernel width should not be larger than the boundary distance, meaning that the gradient of f_h should not surpass 1, which is the dashed line angled at 45° in the plot of Figure 4.7. For several reasons, the gradient should be chosen even lower than 1: If the kernel always extends all the way to the boundary, the particles in the fluid center might have almost all other fluid particles as their neighbor, leading to a high computational cost (Figure 4.8). In addition to that, the boundary distance is computed using a distance propagation method that furthermore only performs one iteration per time step, causing the resulting boundary distance to always be slightly outdated. This boundary distance should only be seen as an approximation that will often over-estimate the distance, so a security margin should be included.

With the linear relationship f_h between boundary distance and kernel width fixed, we can attempt to find a fitting linear relationship $f_{\check{h}}$ between boundary distance and intrinsic kernel width, and finally a linear relationship $f_{\check{r}}$ between boundary distance and target radius. The requirement is, as described in Section 4.2, that every particle's kernel needs to be big enough to cover all adjacent particles, even if they are larger and therefore further away. This means that the intrinsic kernel width $\check{h}_i$ of a particle $\mathbf{p}_i$ can only be chosen in a way where all particles $\mathbf{p}_j$ within the intrinsic kernel (i.e., within range $\check{h}_i$)

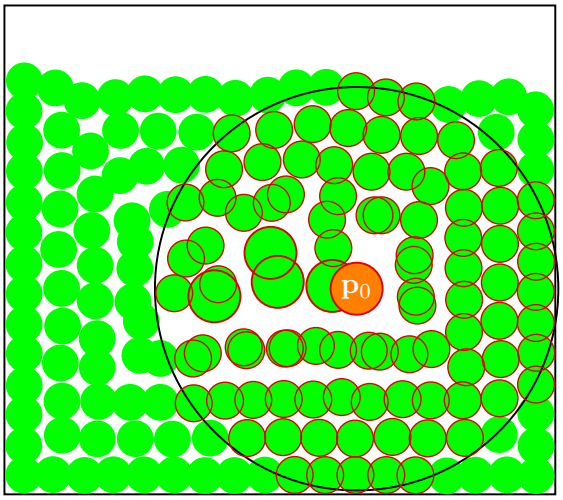


Figure 4.8: The effect of setting the kernel width to the boundary distance: particles near the center of the fluid body might have almost all particles as their neighbors. The kernel centered at $\mathbf{p}_0$ reaches all the way to the boundary, and all neighbor particles within the kernel are highlighted with red circles. The high number of neighbors increases the computational effort.

have a kernel width h_j at least as big as $\check{h}_i$. If we define f_h to be

$$f_h(b) = ab \tag{4.36}$$

with a fixed gradient $a < 1$, then we can construct $f_{\check{h}}$ as

$$f_{\check{h}}(b) = \frac{a}{1+a}b \tag{4.37}$$

so that it fulfills the aforementioned requirement.

Considering our definition of the intrinsic kernel width in Equation 4.35, $f_{\check{r}}$ can easily be derived from $f_{\check{h}}$ as

$$f_{\check{r}}(b) = \frac{a}{k+ka}b. \tag{4.38}$$

Because we specified that the kernel should always lie completely inside the fluid to get an accurate density estimation, the kernel widths (and particles) shrink without limit the closer they get to the boundary. This would result in infinitely many particles, so we define a lower bound $r_{\min}$ for the target radius, which changes Equation 4.38 to

$$f_{\check{r}}(b) = \max\left(\frac{a}{k+ka}b, r_{\min}\right). \tag{4.39}$$

Equations 4.36 and 4.37 change to

$$f_h(b) = \max(ab, kr_{\min}) \tag{4.40}$$

$$f_{\check{h}}(b) = \max\left(\frac{a}{1+a}b, kr_{\min}\right). \tag{4.41}$$

By using $f_{\check{r}}$ and f_h to set the target radius and the kernel width of each particle, the computationally expensive kernel width propagation can be avoided. However, it also introduces a disadvantage: The linear functions increase without limit, and for any given particle $\mathbf{p}_i$, it is predicted that there will be larger adjacent particles further into the fluid, and that the kernel width has to be increased from the intrinsic kernel width to accommodate for that. If $\mathbf{p}_i$ is in the center of the fluid, this kernel width increase is unnecessary and increases the workload because of the larger neighborhood. However, the plots in Chapter 6 show that the impact on the total workload is negligible.

CHAPTER 5

Implementation

In the previous chapter, we discussed several variations of our approach for implementing adaptive sampling in PBF. In this chapter, we describe which of these methods we have selected for our concrete implementation and how they integrate into the general PBD algorithm. Our method selection led to the best results in our tests in terms of quality and performance. The results presented in Chapter 6 have been generated using this setup as well. Furthermore, we also address some implementation details like the used data structures, the data types, and the neighborhood search.

5.1 Algorithm

We implemented the streamlined variant as described in Section 4.6 in a GPU-accelerated manner using C++ and Vulkan 1.2 [Khr21] to exploit the parallelizability of the problem. The steps of the general PBF algorithm extended by our method result in the following sequence of operations:

1. The next particle positions $\mathbf{p}_i$ are predicted based on $\mathbf{x}_i$ and $\mathbf{v}_i$. This is one of the basic PBD steps described in Section 3.1.1.

2. Active particle transfers that perform gradual merges or splits are applied. This step is part of our adaptive sampling extension and is described in Sections 4.3 and 4.4. The active transfers are stored in a GPU buffer where each entry contains references to the source and target particles $\mathbf{p}_s$ and $\mathbf{p}_t$, the remaining duration t_{left} of the transfer, and the transfer type (merge or split). Finished transfers are deleted from the transfer list. Because our final solution uses instant splitting, all splits finish immediately. After a specific gradual merge has finished, its source particle is deleted.

3. Each particle's kernel width is updated using its boundary distance according to the streamlined variant. This step is also added by our method and is described in Section 4.6. As mentioned at the end of Section 4.4.3, the kernel width change per timestep is limited to a certain percentage of the old value to prevent the negative effects of large sudden changes. In our implementation, this limit is set to 1 %.

4. The neighborhood search is executed. Our choice for the neighborhood search algorithm is described in Section 5.4.2. For each particle $\mathbf{p}_i$, all particles $\mathbf{p}_j$ closer than h_i are determined and the neighborhood information is stored in the form of neighbor pairs $(\mathbf{p}_i, \mathbf{p}_j)$.

5. The constraints are solved. This is one of the basic PBD steps described in Section 3.1.1.

6. Each particle's $\nabla_{\mathbf{p}_i} C_i(\mathbf{p})$ that was computed for the incompressibility constraint during the constraint solver's last iteration is used to detect boundary particles. This is one of our added steps and is described in Section 4.5.1.

7. Each particle's boundary distance is updated by performing one iteration of distance propagation. This step introduced by our method is described in Section 4.5.2.

8. Each particle's target radius is computed using the boundary distance. This step, too, is an extension of our method over basic PBD, and is described in Section 4.5.3.

9. Merge and split opportunities are detected and added to the particle transfer list. This step adds the adaptive sampling characteristic to PBF, and is described in Sections 4.5.3 and 4.5.4. For splits, new target particles are generated in the vicinity of the source particles, as described in Section 4.4.1.

10. The particle velocity vectors are updated based on the old and the new particle positions. This is one of the basic PBD steps, as described in Section 3.1.1.

5.2 Particle Data Structure

Allowing particles to split and merge increases the requirements on the used data structures and therefore also the complexity of the implementation. With splitting and merging, the total number of particles fluctuates over time, and particles might have to be deleted from various locations in the buffer, either leading to gaps in the buffer or requiring a memory relocation of other particles to fill the gap. As our implementation runs on the GPU, we often execute one thread per element in the particle buffer. If this buffer were to contain gaps, the respective threads would have no work to do, leading to high divergence within warps and reducing the efficiency. Therefore, it is preferable to fill the gaps. In case the particles are resorted every timestep to preserve their locality[1], the particles have to be moved anyway, which can be combined with filling the gaps.

[1] For example by using a Z-order curve [Mor66]; the particles are stored in the buffer in a way so that particles with positions close to each other are also likely stored close to each other within the buffer.

The memory relocation of particles, however, gives rise to another problem: Our merging processes are happening over a period of time during which we need to keep references to the source and the target particles which are currently in the process of being merged. Whenever a particle is relocated within the buffer, its reference has to be updated as well.

This could be avoided by alternating between two phases: One where merges are happening, and one immediately after the merges finished, where the gaps caused by the merges are closed. Particle resorting for locality preservation could also only happen in the latter phase. As this limits the times when merging can happen, in our implementation we are following the reference-update approach instead.

5.3 Data Type of Particle Positions

Our GPU implementation solves multiple constraints simultaneously. Several of these constraints might share the same particle as being of influence, and they might attempt to shift this particle at the same time, possibly in different directions. To handle this concurrent memory access, we use atomic add functions provided by the GPU. The Vulkan API—which we used for the GPU implementation—only supported atomic add functions for integer data types at the time when we implemented our solution [Khr21]. Therefore, we decided to store the particle positions not as float vectors, but as integer vectors, and scale the entire simulation so that the resolution of the position grid is sufficiently fine.

There is another argument that can be made for choosing integer over float for representing positions: The float data type is especially precise around the value zero, but loses precision with increasing absolute value. This makes it especially useful for representing velocities or masses, where small value changes are of more importance if they happen close to zero than if they appear at some high value: As an example, the difference between masses of 0.01 and 0.11 is very important, while the difference between masses of 100.0 and 100.1 is most of the time negligible, even though in both cases the difference is 0.1. However, in the case of positions, value changes are of equal importance across the whole domain. Moving an object by a distance of 0.1 has the same relevance at position 0.01 as it has at 100.0. With integer values, the precision is equal everywhere within the representable range, which makes them appropriate for storing positions.

5.4 Neighborhood Search

5.4.1 Grid-Based Neighborhood Search

Finding the neighbor particles to each fluid particle makes up a large portion of the necessary computation time per timestep. A commonly used method in the field of fluid simulation is the grid-based neighborhood search [MM13, MMCK14]. An efficient GPU implementation was first described by Green [Gre08]. For this method, a uniform grid spanning the whole simulation domain is used. Each particle is assigned to a cell, and

the grid data structure is set up to allow the lookup of cell content in constant time. Using this data structure, finding the neighbors of a particle $\mathbf{p}_i$ amounts to the lookup of all cells close to the cell that $\mathbf{p}_i$ belongs to.

In the original method, Green [Gre08] recommends to set the cell size to be equal to the particle size $2r$ to find particle collisions. For incompressibility constraints, the cell size accordingly should be equal to the kernel width h to find all neighbors. With this condition fulfilled, only the directly adjacent cells (including diagonal adjacency) have to be regarded, leading to a lookup of 3^d cells (9 cells in 2D or 27 cells in 3D).

Our fluid simulation method leads to particles of varying sizes and kernel widths, which makes this recommended optimization impossible. Setting the cell size according to the smallest occurring particle radius $r_{\min}$ to $kr_{\min}$ has the effect that larger particles have to lookup a large number of neighbor cells. On the other hand, setting the cell size to a larger value leads to an insufficient resolution in the area of small particles so that each cell contains a large amount of particles, many of which will not be close enough to be relevant for the neighborhood search.

The second argument against the grid-based approach for the neighborhood search is that it requires to restrict the simulation domain to an area small enough to be covered by a uniform grid. To allow for linear cell lookup times, the data structure requires storage proportional to the number of cells. Therefore, the possible size of the grid is severely limited. A grid with the dimensions $2048 \times 2048 \times 1024$ cells already exhausts the whole range of numbers representable by an unsigned 32-bit integer for enumerating the cells. One advantage of Lagrangian fluid simulation over Eulerian fluid simulation is that the simulation domain does not have to be limited by a fixed-size grid, and using a fixed-size grid for the neighborhood search removes that advantage. The core of this problem is that the density of particles will likely vary throughout the simulation domain. Some regions do not contain any fluid at all, and with our variable sampling method, some regions deep within the fluid have a coarser fluid resolution and therefore contain fewer particles. As the cells are only sparsely occupied, a possible countermeasure is to use a hash map to store the cell contents. This preserves the linear cell lookup time but reduces the required storage. However, this also introduces a new disadvantage by increasing the randomness of the storage access.

5.4.2 Binary Neighborhood Search

This section describes our modifications to the grid-based neighborhood search by Green that avoid the two problems described above (fixed cell size and simulation domain bounded by grid size), while also avoiding scattered storage access. This method is related to octrees and uses the Z-order curve in combination with binary searches to find the neighbors.

The Z-order [Mor66, TH81] assigns a scalar value to any multidimensional data point in a way so that two data points that are close together in their multidimensional space are likely also close together in their one-dimensional mapping. This makes the Z-order

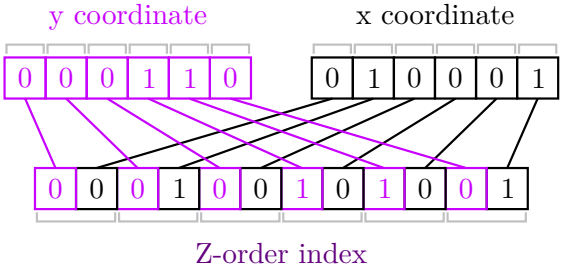


Figure 5.1: For the Z-order, the binary representations of a cell's coordinates are interlinked. In this example, the Z-order index of the two-dimensional point $(17, 6)$ is computed. This Z-order index can also be seen as a sequence of quadtree node IDs. From left to right, each digit pair equals the next child index, beginning from the tree root: 0, 1, 0, 2, 2, 1.

very useful for storing the particles, because then neighboring particles are mostly close together in memory, which improves the performance of buffer reads. The Z-order bijectively maps cells of a uniform grid to a unique integer. The uniform grid has d dimensions, and for every cell, each of its d coordinates is stored in the form of a binary number with g digits. Because it is bijective (one-to-one correspondence), the mapped integer has $d \cdot g$ digits in its binary representation.

The mapping works by interlinking the binary digits of the d coordinates (Figure 5.1 shows an example with $d = 2$). The least significant bit of every coordinate is placed in the d least significant bits of the resulting integer, the next d digits of the result are the next-to-least binary digits of every coordinate, and so on. As a result, leaving out the last $\bar{g} \cdot d$ digits of the Z-order index is the same as leaving out the last $\bar{g}$ digits of every cell coordinate, which in turn is the same as reducing the grid's granularity. In fact, the Z-order is closely related to quadtrees and octrees [Mor66]: In an octree (where $d = 3$), the first three binary digits of the Z-order index identify the child of the octree root node that contains the data point. The next three digits identify this child's child node, and so on.

This property simplifies the access to the data on different coarseness levels: If the Z-order was computed based on a $2^g \times 2^g \times 2^g$ uniform grid, then we can easily interpret the data as being structured in a coarser grid of $2^{g-1} \times 2^{g-1} \times 2^{g-1}$ just by omitting the last three digits of the Z-order index. Any of the g granularity levels is accessible by dropping a multiple of three digits from the end of the index.

In Section 5.4.1, we briefly describe the grid-based neighborhood search by Green [Gre08], but also explain why we consider this method unfitting for our implementation. One of the reasons is the uniformness of the grid, which is less suitable if the particle density—and especially the queried neighborhood area—varies depending on the location. This problem is avoided now, because as described above, the Z-order can be used to interpret the data as being structured in different grid resolutions. Our other argument against using the grid-based neighborhood search was in regard to the limitation of the overall

size of the uniform grid. Because each cell needs its own space in memory, the number of cells is limited by the available memory. In our method, we instead store the particles in a list sorted in Z-order, but without leaving a space in the place of empty cells. Due to this, we cannot achieve the constant lookup time that is possible with the grid-based method by Green [Gre08]. Instead, we use binary search to look for a Z-order index. As we store particle positions as 32-bit integer vectors, the data points are already in a uniform grid with a size of 2^{32} cells along each axis. For a three-dimensional simulation, the Z-order index accordingly has to be a 96-bit unsigned integer, so that the grid covers the whole available space.

The final algorithm for our neighborhood query based on binary search is as follows:

1. Calculate the Z-order index of every particle.
2. Sort the particles by their Z-order index.
3. For every particle $\mathbf{p}_i$, find its neighbors within a certain particle-dependent distance h_i:
   a) Select the finest grid resolution where the cell widths are not smaller than h_i.
   b) Omit the corresponding number of binary digits from the end of the Z-order index in all following computations, so that we work in the correct grid resolution.
   c) Get the Z-order index of $\mathbf{p}_i$.
   d) Get the Z-order indices z_n of all 3^d neighboring cells (which includes the cell $\mathbf{p}_i$ is located in).
   e) Find all particles with a Z-order index equal to any z_n by using 3^d binary searches on the sorted particle list.
   f) Remove all particles with a distance to $\mathbf{p}_i$ greater than h_i from the neighborhood query result.

5.4.3 Ray Tracing-Based Neighborhood Search

As an alternative to the binary neighborhood search, we tested one more approach that is currently only applicable for a small subset of available graphics cards. For this method, we exploit the ray tracing technology that is supported by some of the latest high-end graphics cards, like NVIDIA's RTX GPUs [NVI18]. The following description assumes a three-dimensional fluid simulation, but applies in the same way to one- or two-dimensional cases.

The ray tracing API allows us to create a list of bounding boxes that represent the scene, build an acceleration structure (we are using the Vulkan extension `VK_KHR_acceleration_structure` [Khr21]), and then query all boxes that intersect with a given line segment using `VK_KHR_ray_query`. This line segment can be

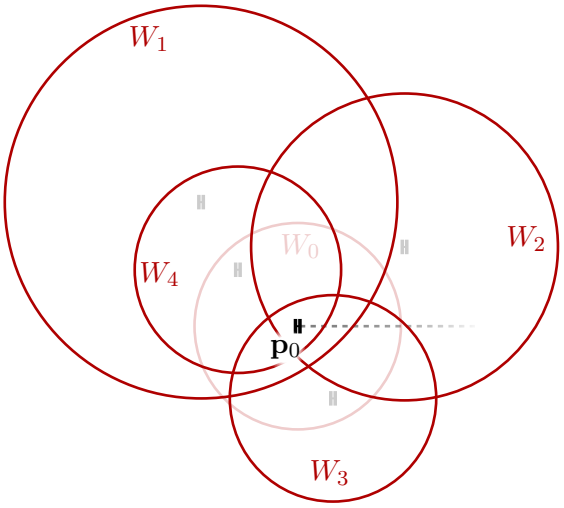


Figure 5.2: A ray tracing query used for the neighborhood search. A ray starting from position $\mathbf{p}_0$ is created and set to a short length so that it approximates a single point. Using this ray, the ray tracing API will report hits on the objects (kernels) W_0, W_1, W_2, W_3, and W_4. From this, we know that $\mathbf{p}_0$ is a neighbor of $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and $\mathbf{p}_4$. It is worth noting that the inverse is not necessarily true: $\mathbf{p}_1$ and $\mathbf{p}_2$ are *not* neighbors of $\mathbf{p}_0$.

set to a very short length, which effectively turns the query from a line-box intersection into a point-box intersection. By defining the box as the bounding box of a sphere and filtering out any query results where the "point" lies outside of the sphere, the query can be turned into a point-sphere intersection.

For our fluid simulation method, we need to find all neighbor particles that lie within the kernel W_i for each particle $\mathbf{p}_i$. Specifically, for each particle $\mathbf{p}_i$, we need to query all particles within a sphere of radius h_i that is centered at $\mathbf{p}_i$. This is equivalent to a point-sphere intersection and can be done using the ray tracing API.

The data passed to the ray tracing API for building an acceleration structure are the bounding boxes of all n kernels. Afterwards, we perform a ray tracing query for each particle $\mathbf{p}_j$, where the ray is a very short line segment at position $\mathbf{p}_j$. Figure 5.2 shows one such query. The result of the query is a list of kernels which encompass $\mathbf{p}_j$, meaning that the result is not the list of neighbors, but a list of particles $\mathbf{p}_i$ that $\mathbf{p}_j$ *is a neighbor of*. To get the list of neighbors for each particle $\mathbf{p}_i$, we could append $\mathbf{p}_j$ to the neighbor list of each query result $\mathbf{p}_i$. However, as we do not need the lists explicitly compiled, we instead append the information about the neighborhood of $\mathbf{p}_j$ to $\mathbf{p}_i$ to a list of all neighborhood relationships within the fluid.

5.5 Challenges and Remaining Problems

An inherent problem of our streamlined variant is that the kernel width increase happens before merging. This causes the neighbor pair count to initially increase due to the larger kernels before being reduced due to the overall particle count reduction after merging has finished.

Another inconvenience is caused by the fact that we do not impose any restrictions on allowed particle radii with the goal to merge and split as often as possible. This can lead to situations where a particle gets near to the boundary with a radius bigger than the minimum target radius, but the particle is too small to be split. Such a situation can occur, for example, if three particles of minimum radius (called *minimum particles* in the context of this example) merge together. After one split of this big particle, the two resulting particles are still bigger than the minimum radius, but neither of them can split any further, since both only consist of 1.5 minimum particles. More generally, this problem occurs whenever a particle that does not consist of an even number of minimum particles gets split. One way to avoid this would be to only allow merges of two equally sized particles, but this could prevent many useful merges. Alternatively, splits of particles containing an uneven number of minimum particles could be implemented in a way so that the resulting two particles do not contain fractions of minimum particles by not having them both equally sized.

CHAPTER 6

Results

In this chapter, we compare our method (using the streamlined variant) with basic PBF where all particles have the same mass and kernel width. We compare two scenes, each of them in both 2D and 3D: a waterfall scene where the fluid starts in the upper pool and pours down into the lower pool, and a circular/spherical fluid body without any movement, which is the ideal setup to showcase the advantages of our method. For the 2D and 3D waterfall scene, we compare the runtime of our method with the runtime of basic PBF. The most time-consuming steps are the neighborhood search and the constraint solving, so we list the durations of these two steps in addition to the overall time it takes to complete an entire simulation step. Because the computation time of our method fluctuates, we capture it at several points in time instead of averaging it over the whole simulation to present more meaningful results.

The runtime fluctuations in our method were expected: The number of compute shader threads for the neighbor search depends on the particle count, and the number of threads for constraint solving is determined by the particle count as well as the neighbor pair count. Both of these counts greatly vary throughout the simulation with our method—the particle count due to merging and splitting, and the neighbor count due to the variations in sampling density, kernel width, and particle count. Regarding the particle count, our method provides certain guarantees: With sufficient fluid depth, our method will always reduce the number of particles. It is also guaranteed to never *increase* the number of particles. For the neighbor pair count, no such general statements can be made. Therefore, the analysis in this chapter will mainly focus on the number of neighbor pairs in different scenarios.

Due to the increasing kernel widths, the number of neighbor pairs can be higher with our method compared to basic PBF. This is especially noticeable in situations where a big fluid body consisting only of small particles starts the merging process—this situation might occur immediately after initialization of a fluid body, or after strong turbulence inside the fluid that caused most particles being split into particles of smallest size. Our method first

increases the kernel width, leading to an increase of neighbor pairs, followed by particle merges, leading to a reduction of particles and neighbor pairs. Whether the reduction of neighbor pairs as a result of the merges manages to outweigh the initial increase depends on the fluid depth. Section 6.1 addresses this question with a mathematical analysis, while Section 6.2 presents actual measurements from our simulations.

6.1 Expected Reduction of the Number of Neighbor Pairs

In this section, we analyze the expected number of neighbor pairs when using our method—the streamlined variant described in Section 4.6—and compare it to the expected number of neighbor pairs when using basic PBF where all particles represent the same amount of fluid. We do these computations for the one-, two-, and three-dimensional case and compare these results to see the effect of the dimensionality on the results.

For simplicity, we assume a spherical/circular fluid body with the radius r_f, centered at the origin of the coordinate system. The distance to the fluid boundary is therefore defined as

$$b = r_\mathrm{f} - |x|, \tag{6.1}$$

with x being the distance from the fluid center.

After horizontally shifting the function so that the fluid center is at $x = 0$, the computation of the kernel width turns from Equation 4.40 into

$$f_h(x) = \max(a(r_\mathrm{f} - |x|), kr_\mathrm{min}). \tag{6.2}$$

The target radius is defined similar to Equation 4.39, only shifted:

$$f_{\check{r}}(x) = \max\left(\frac{a}{k + ka}(r_\mathrm{f} - |x|), r_\mathrm{min}\right). \tag{6.3}$$

Equations 6.1 to 6.3 apply to one-, two-, and also three-dimensional setups. The following equations are specifically for the one-dimensional case: Assuming that the particle radii match the target radii, the particle density can be computed using

$$f_\mathrm{d}(x) = \frac{1}{2f_{\check{r}}(x)}. \tag{6.4}$$

The expected number of neighbors of a particle at any given distance from the fluid center can be computed with

$$f_\mathrm{n}(x) = \int_{x-f_h(x)}^{x+f_h(x)} f_\mathrm{d}(y_1)\,\mathrm{d}y_1. \tag{6.5}$$

Weighing this number by the particle density gives us the neighbor pair distribution

$$f_\mathrm{p}(x) = f_\mathrm{d}(x) f_\mathrm{n}(x), \tag{6.6}$$

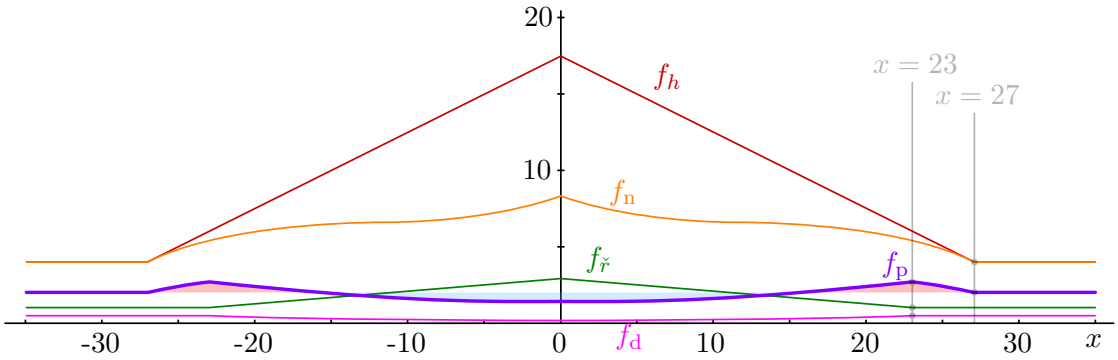


Figure 6.1: The distribution of neighbor pairs within a one-dimensional fluid of radius 35. This plot shows the functions from the Equations 6.2 to 6.6. f_h represents the kernel width, which has a lower bound of $kr_\mathrm{min} = 4$ and starts increasing at a boundary distance of $b = \frac{kr_\mathrm{min}}{a} = 8$ ($x = 27$ from the fluid center). With the growth of the kernel, the neighbor count f_n also increases, but reaches an equilibrium as soon as the kernel only contains the linear section of $f_{\check{r}}$ (between $x = 0$ and $x = 23$). Closer to $x = 0$, the kernel overlaps with the peak of $f_{\check{r}}$, again leading to an increase of the neighbor count. This increase of neighbors per particle is counteracted by the reduction of particles starting at $x = 23$. The particle density f_d shrinks with increasing fluid depth, resulting in the actual neighbor pair distribution f_p. Compared to non-adaptive PBF, the outskirts of the fluid produce more neighbor pairs (light red area), but the center of the fluid produces less (light blue area). With increasing fluid radius r_f, the blue area outweighs the red one, leading to fewer neighbor pairs, which in turn means there are less computations necessary.

and integrating this over the whole fluid body results in the expected number of neighbor pairs:

$$p_\mathrm{e} = \int_{-r_\mathrm{f}}^{r_\mathrm{f}} f_\mathrm{p}(y_1)\,\mathrm{d}y_1. \tag{6.7}$$

Figure 6.1 shows plots of these functions for $r_\mathrm{f} = 35$, $a = 0.5$, $k = 4$, and $r_\mathrm{min} = 1$. For small fluid bodies, our method causes the number of neighbor pairs to be higher than in basic PBF. The kernels of many particles grow in size, but due to the lack of fluid depth, not many particles get merged. The small reduction of the number of particles is outweighed by the increase of neighbors per particle, so that our method is inferior for shallow fluids. However, with increasing depth, more particles get merged, which soon mitigates the effects of the increased kernel size. With the properties

$$\begin{aligned} a &= 0.5 \\ k &= 4 \\ r_\mathrm{min} &= 1, \end{aligned}$$

our method starts reducing the number of neighbor pairs compared to basic PBF at $r_\mathrm{f} \approx 32$ (Table 6.1).

	$a = 0.5$	Fixed a From Table 6.2
$d = 1$	32	32
$d = 2$	50	47
$d = 3$	80	62

Table 6.1: The minimum fluid body radius r_f where our method starts to produce fewer neighbor pairs than basic PBF. The other properties are fixed to $k = 4$ and $r_\mathrm{min} = 1$. With increasing dimensionality, the required radius also increases.

In two dimensions, the Equations 6.4 to 6.7 become

$$f_\mathrm{d}(x) = \left(\frac{1}{2f_{\check{r}}(x)}\right)^2 \tag{6.8}$$

$$f_\mathrm{n}(x) = \int_{-f_h(x)}^{f_h(x)} \int_{-\sqrt{f_h(x)^2 - {y_1}^2}}^{\sqrt{f_h(x)^2 - {y_1}^2}} f_\mathrm{d}\left(\sqrt{(x + y_1)^2 + {y_2}^2}\right) \mathrm{d}y_2 \, \mathrm{d}y_1 \tag{6.9}$$

$$f_\mathrm{p}(x) = f_\mathrm{d}(x) f_\mathrm{n}(x) \tag{6.10}$$

$$p_\mathrm{e} = \int_0^{2\pi} \int_0^{r_\mathrm{f}} f_\mathrm{p}(y_2) y_2 \, \mathrm{d}y_2 \, \mathrm{d}y_1. \tag{6.11}$$

With the same values for a, k, and r_min as above, the turning point from which on our method produces fewer neighbor pairs is at $r_\mathrm{f} \approx 50$ (Table 6.1).

The equations for the three-dimensional case are

$$f_\mathrm{d}(x) = \left(\frac{1}{2f_{\check{r}}(x)}\right)^3 \tag{6.12}$$

$$f_\mathrm{n}(x) = \int_{-f_h(x)}^{f_h(x)} \int_{-\sqrt{f_h(x)^2 - {y_1}^2}}^{\sqrt{f_h(x)^2 - {y_1}^2}} \int_{-\sqrt{f_h(x)^2 - {y_1}^2 - {y_2}^2}}^{\sqrt{f_h(x)^2 - {y_1}^2 - {y_2}^2}} f_\mathrm{d}\left(\sqrt{(x + y_1)^2 + {y_2}^2 + {y_3}^2}\right) \mathrm{d}y_3 \, \mathrm{d}y_2 \, \mathrm{d}y_1 \tag{6.13}$$

$$f_\mathrm{p}(x) = f_\mathrm{d}(x) f_\mathrm{n}(x) \tag{6.14}$$

$$p_\mathrm{e} = \int_0^{2\pi} \int_0^{\pi} \int_0^{r_\mathrm{f}} f_\mathrm{p}(y_3) {y_3}^2 \sin(\mathrm{d}y_2) \, \mathrm{d}y_3 \, \mathrm{d}y_2 \, \mathrm{d}y_1. \tag{6.15}$$

Using the same values for a, k, and r_min also in this case, our method starts producing better results at $r_\mathrm{f} \approx 80$ (Table 6.1).

While we have constantly used $a = 0.5$ for our analysis so far in order to directly compare the performance of our method for different dimensionalities, the optimal value for a actually varies depending on the dimensionality and the other properties $(r_\mathrm{f}, k, r_\mathrm{min})$.

We do not consider variations for k, as smaller values would lead to wrong density estimates, and larger values are guaranteed to increase the number of neighbor pairs

	$r_\mathrm{f} = 35$	$r_\mathrm{f} = 50$	$r_\mathrm{f} = 75$	$r_\mathrm{f} = 100$	$r_\mathrm{f} = 200$	Fixed
$d = 1$	0.44	0.47	0.5	0.5	0.51	0.5
$d = 2$	0	0.33	0.4	0.4	0.44	0.4
$d = 3$	0	0	0.28	0.31	0.35	0.3

Table 6.2: The optimal values of a for each dimensionality, depending on the fluid body radius r_f. The rightmost column contains our chosen fixed values for each dimensionality.

	Particle Count			Neighbor Pair Count		
	Basic	Our Method	Reduction	Basic	Our Method	Reduction
$d = 1$	100	38	−62.0 %	400	218	−45.5 %
$d = 2$	7854	3331	−57.6 %	98 696	63 245	−35.9 %
$d = 3$	523 599	296 159	−43.4 %	17 545 963	13 809 794	−21.3 %

Table 6.3: A comparison of the expected number of particles and neighbor pairs when using basic PBF and when using our method. The fluid radius r_f is set to 100, r_min is set to 1, k is 4, and the fixed values from Table 6.2 are used for a.

and worsen the performance. Without loss of generality, we also fix $r_\mathrm{min} = 1$. Doubling r_min is the same as halving r_f, meaning the relation between r_min and r_f is what actually matters. This leaves r_f as the only remaining influence to consider.

Table 6.2 lists approximations of the ideal a values (in regard to the resulting neighbor pair count) in the one-, two-, and three-dimensional case for a selection of fluid body sizes. These values show that the dimensionality has a rather strong influence on the ideal a values, while the different r_f values only lead to minor variations (once the fluid surpasses the size given in Table 6.1). Consequently, we decided that using a fixed global value for a that only depends on the dimensionality is sufficient. Our values chosen as the "fixed a" are listed in Table 6.2 and are based on the other optimal values for a, also listed in the table.

Using these fixed values for a, the thresholds for r_f from which on our method reduces the total number of neighbor pairs shrink to $r_\mathrm{f} = 47$ and $r_\mathrm{f} = 62$ for two and three dimensions, respectively (see Table 6.1). The actually expected reduction in the particle count and the neighbor pair count for a fluid of radius 100 is listed in Table 6.3. Figure 6.2 and Figure 6.3 plot the expected reduction in the particle count and the neighbor pair count depending on the size of the fluid body.

One thing to take note of is that with our method, the number of neighbors $f_\mathrm{n}(x)$ strongly varies between all fluid particles. Figure 6.4 shows that especially in higher dimensions, particles in the center of the fluid can have over four times more neighbors compared to particles at the fluid boundary. Depending on the implementation of the incompressibility constraint solver, this might introduce additional computational bottlenecks. If the solver

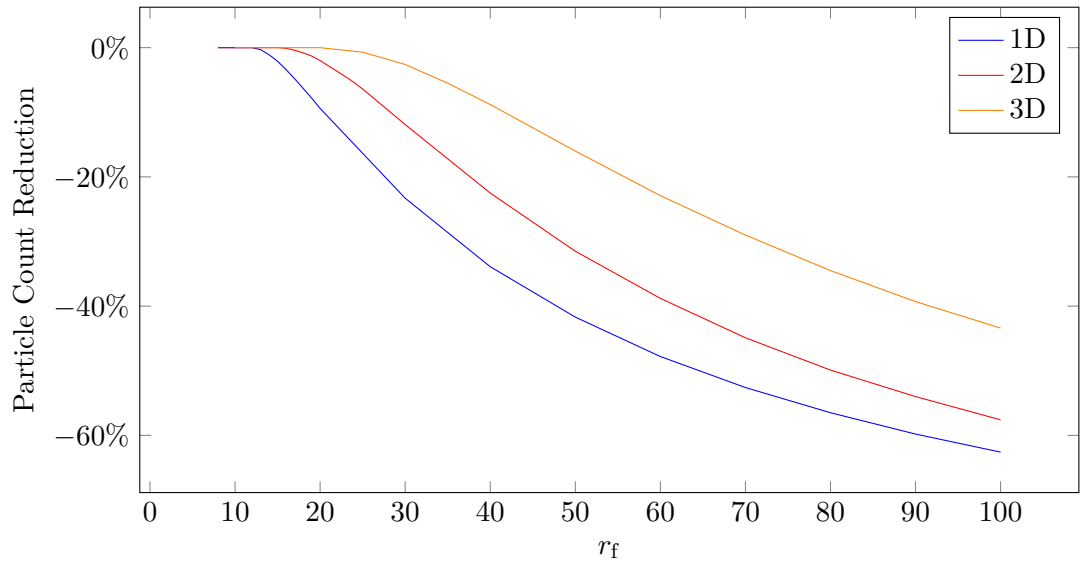


Figure 6.2: The expected particle count reduction for different fluid sizes. The fixed values from Table 6.2 are used for a in each respective dimensionality. The other properties are fixed to $k = 4$ and $r_{\min} = 1$.

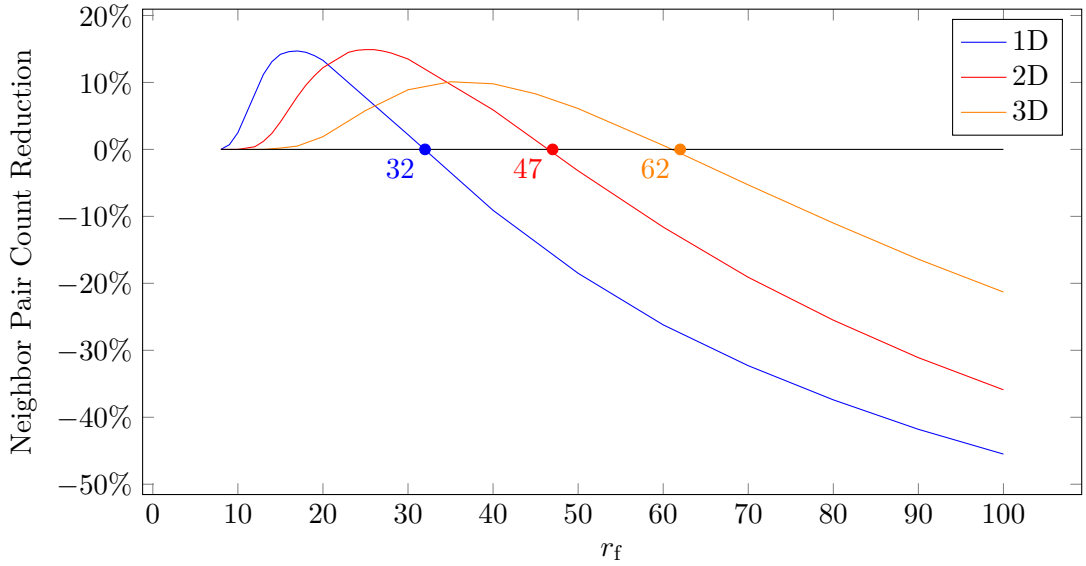
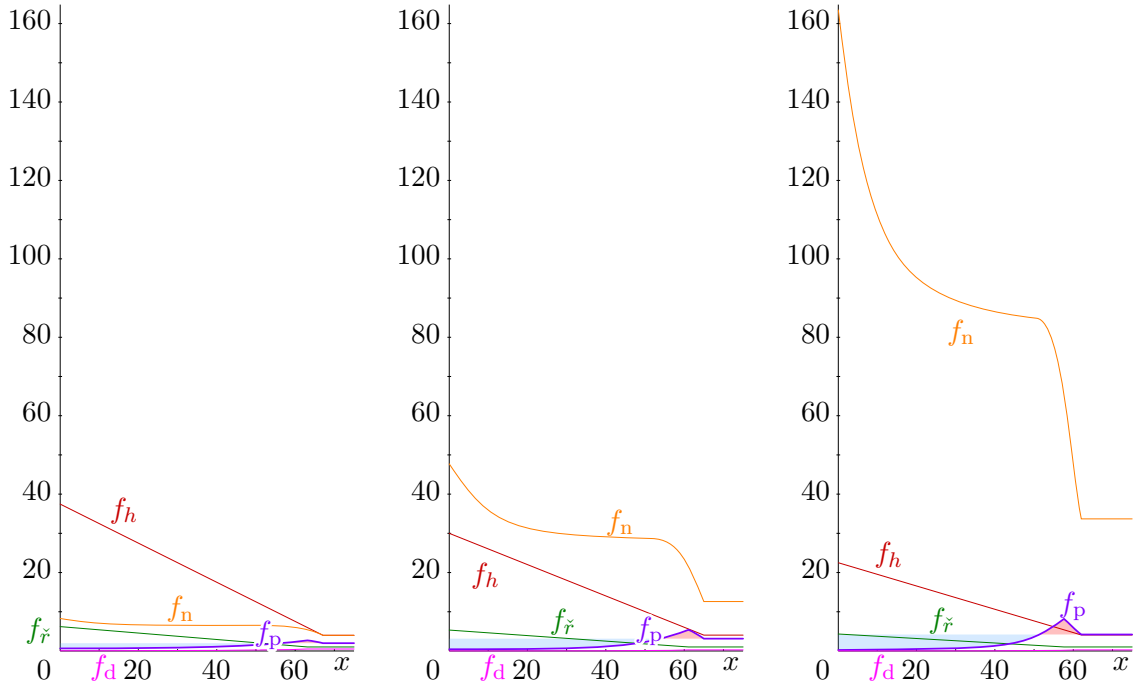


Figure 6.3: The expected neighbor pair count reduction for different fluid sizes. The fixed values from Table 6.2 are used for a in each respective dimensionality. The other properties are fixed to $k = 4$ and $r_{\min} = 1$. The 0 % crossing points result in the values listed in Table 6.1.



(a) $d = 1$ (b) $d = 2$ (c) $d = 3$

Figure 6.4: The dimensionality has high influence on the shape of the functions used to predict the number of neighbor pairs. With an increasing number of dimensions, the number of neighbors per particle $f_n(x)$ increases dramatically—especially in the center of the fluid. All three plots use the properties $r_f = 75$, $k = 4$, and $r_{min} = 1$. For a, the fixed values from Table 6.2 are used.

should be parallelized on the GPU, then one possible approach might be to start one thread per fluid particle and in each thread iterate over all of the particle's neighbors to use them in the necessary calculations. Because of the high fluctuation of the neighbor count, there is a high potential for loop divergence, which will likely incur performance penalties.

Having the particles stored in a locality preserving order (e.g., the Z-order [Mor66]) can reduce the severity of this problem. The number of neighbors is similar for particles that are close together ($f_n(x)$ is continuous), so if the particles handled within the same warp are close together, the loop divergence is minimized.

Alternatively, the solver can be implemented using a different structure: Instead of having one thread per particle, each thread could handle one neighbor pair. This ensures that each thread has the same workload. However, this method also severely increases the number of threads and necessary memory reads.

In our test implementation, we decided on the latter option in conjunction with atomic add operations. While the thread divergence is kept low, the computation time of our

test implementation is rather high, probably due to the high number of memory reads and atomic operations.

6.2 Measurements

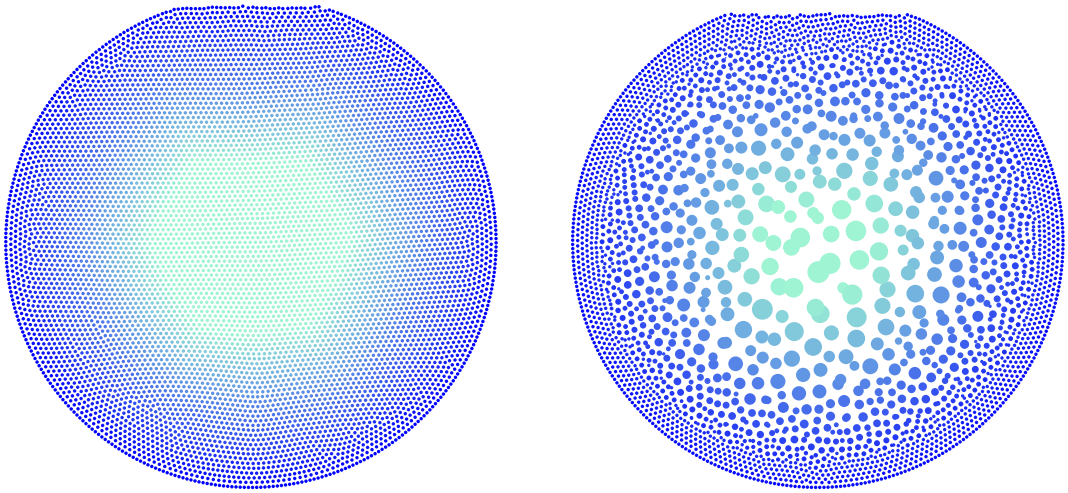
In this section we will present some measurements of our test implementation to evaluate our method. Additionally, we will show some screenshots that allow a visual comparison of our method with the original PBF. Finally, we also include some measurements that evaluate the accuracy of the equations presented in Section 6.1.

As mentioned before, our test implementation is currently not optimized for performance: We use an indexed data structure for the particles that allows for a modular design where fluid particles can be stored as regular particles with their fluid-specific properties being stored in separate dedicated buffers, but the extensive use of indexed data structures also adds some overhead in computation and especially memory access. Optimizing the performance of our implementation would require major structural changes, which are left for future research.

Nonetheless, the neighbor pair count reduction discussed in Section 6.1 shows the advantages of our method over the original PBF for large fluid bodies. Figure 6.5 compares a circular fluid simulated with basic PBF to the same circular fluid simulated with our method. Some particle properties of the fluid in Figure 6.5b are plotted in Figure 6.6 and compared to the expected properties based on the equations from Section 6.1. As this circular fluid has the same properties as the one analyzed in Table 6.3, the particle count and the neighbor pair count can also be directly compared to the expected values, which is done in Table 6.4.

For a comparison of our method to basic PBF in regard to the visual results, we have listed several snapshots of the two simulations in Figure 6.7. In this two-dimensional scenario, the fluid pours from one container down into another. Since most of the time the fluid is splashing and swirling, our method keeps the particle size small throughout the pouring phase. Once the fluid gathers in the lower container, the particles start merging again. However, in preparation for merging, the kernel widths increase beforehand, leading to an increased number of neighbor pairs. After reaching a peak of 231 567 neighbor pairs in Figure 6.7l, the particle reduction due to the ongoing merges takes effect and reduces the number of neighbor pairs in Figures 6.7n and 6.7p.

The measured computation times listed in Table 6.5 reflect the numbers of particles and neighbor pairs: The lower number of particles in our method reduces the time spent searching each particle's neighbors, and the increased number of neighbor pairs during the start of merging leads to more time spent solving incompressibility constraints. In the first snapshot after 30 seconds, the neighbor pair count is reduced by a quarter, but the constraint solving still takes approximately the same time. This might be caused by our test implementation using atomic operations for shifting the particles: In our method, most particles are neighbor to more particles than in basic PBF, resulting in



(a) Basic PBF (b) Our Method

Figure 6.5: A comparison between basic PBF and our method on the example of a circular, two-dimensional fluid. In (a) are 7853 particles, while in (b), our method has reduced the number of particles to 3154.

	Particle Count			Neighbor Pair Count		
	Basic	Our Method	Reduction	Basic	Our Method	Reduction
Expected	7854	3331	−57.6 %	98 696	63 245	−35.9 %
Measured	7853	3154	−59.8 %	92 200	66 011	−28.4 %

Table 6.4: A comparison of the expected particle count and neighbor pair count listed in Table 6.3 to the numbers measured from our simulated fluid depicted in the screenshots in Figure 6.5. Our method actually reduced the number of particles more than expected. This is probably caused by bigger particles advancing towards the boundary but still not being close enough to perform a split, or also by particles affected by the problem described in Section 5.5. On the other hand, the total neighbor pair count shows less reduction than expected, which aligns with the results from Figure 6.6, as this plot shows that the neighbor count is higher than expected for all particles closer to the fluid center. Lastly, the difference between the expected and the actual neighbor pair count for both methods might be surprising, but this is probably caused by Equation 6.8 not accounting for the missing particles beyond the fluid boundary.

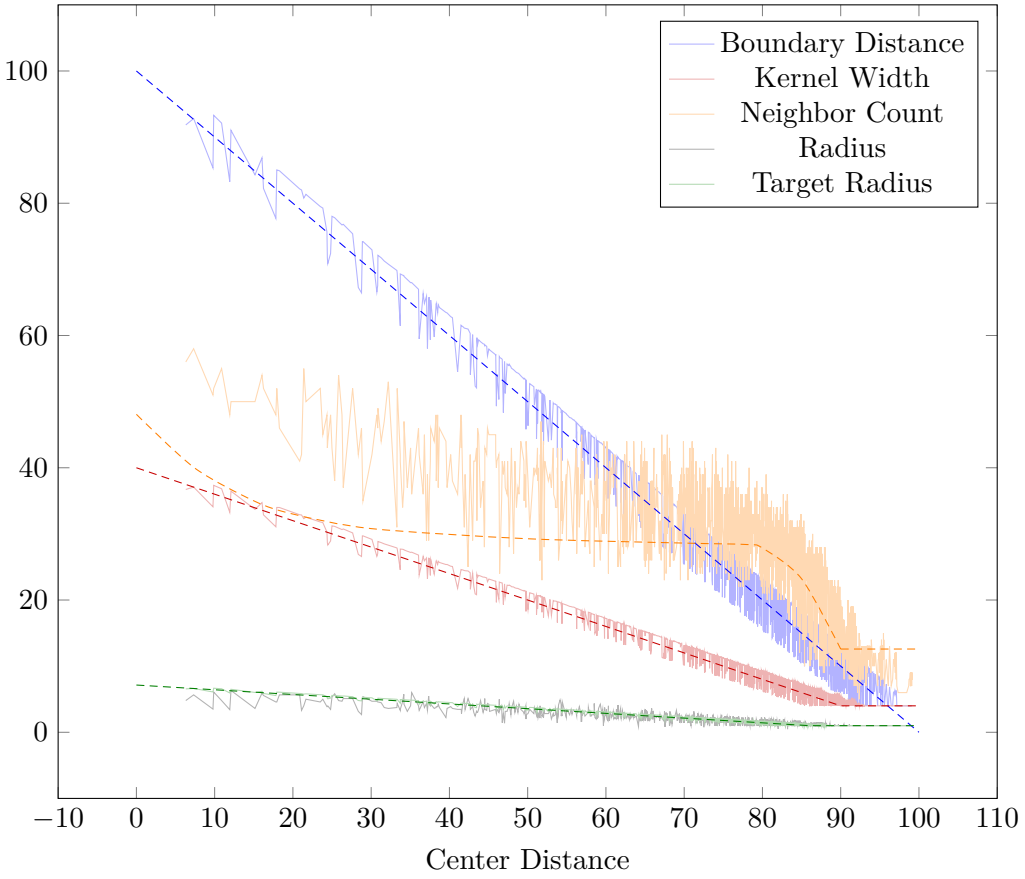
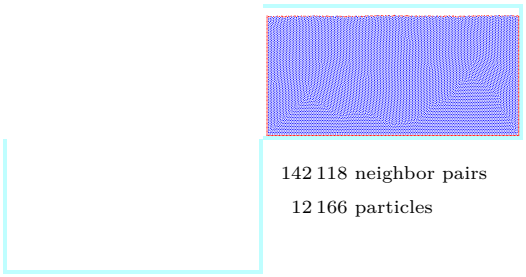


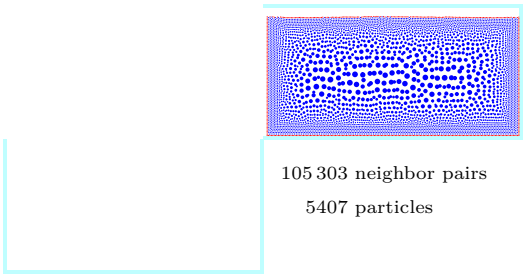
Figure 6.6: A selection of particle properties measured from the two-dimensional circular fluid depicted in Figure 6.5b is plotted against the particle distance from the fluid center. The expected values are drawn as dashed lines, while the actual values form the zigzag lines. The boundary distance based on propagation starting at the boundary is close to the expected value throughout the fluid, and so are the directly derived kernel width and target radius. Merging and splitting are mostly able to closely adapt the actual radius to the target radius. Only near the fluid center, the radius lies below the target radius because some merge opportunities were not yet detected, or the merges were not finished. The neighbor count is higher than expected, probably due to particle clustering and the variation in particle sizes caused by ongoing or missed merges. Nevertheless, the total neighbor pair count is with 66 011 noticeably smaller than with basic PBF, where the neighbor pair count is 92 200.

increased simultaneous write accesses to the same memory location when each constraint shifts the neighbor particles.

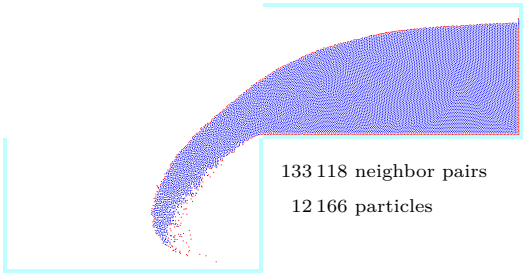
A similar comparison, but for three dimensions, is done in Figure 6.8, with the corresponding computation times listed in Table 6.6. With the higher dimensionality, the target radius grows even slower with increasing boundary distance. The fluid is not very deep, so only few merges were performed in the first 30 seconds. Instead of showcasing the merging and splitting, this comparison mainly shows that the kernel width variation in our method has no noticeable adverse effect on the visual simulation results. In its current state, our method does not provide any advantage when applied to fluids of insufficient depth like in this example. The reason for this limitation is that our method sets all boundary particles to the smallest possible size and should be addressed in future research.



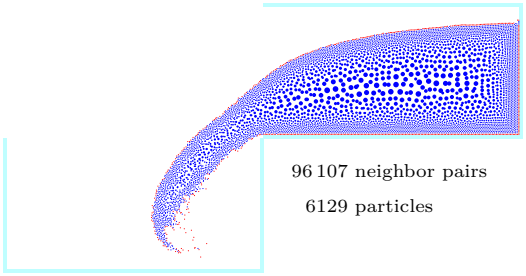
(a) Basic PBF after 30 seconds



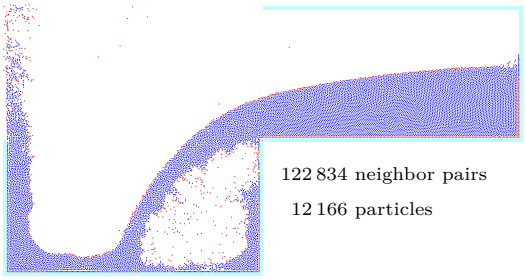
(b) Our method after 30 seconds



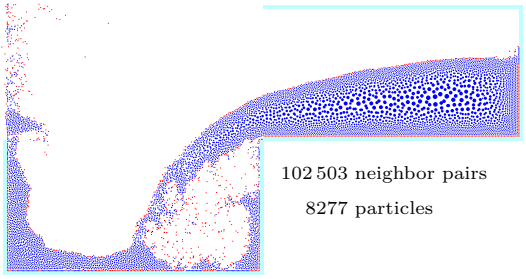
(c) Basic PBF after 34 seconds



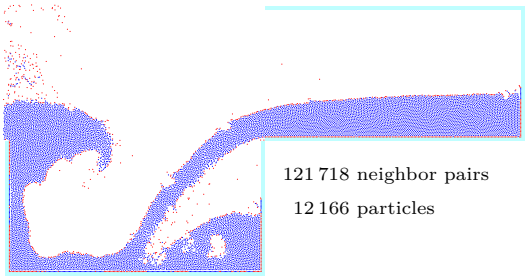
(d) Our method after 34 seconds



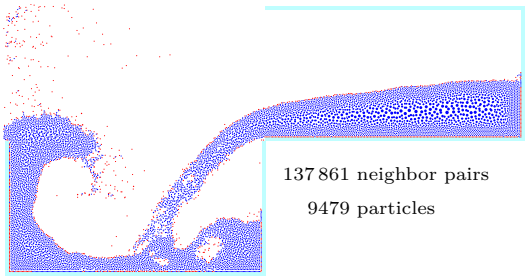
(e) Basic PBF after 40 seconds



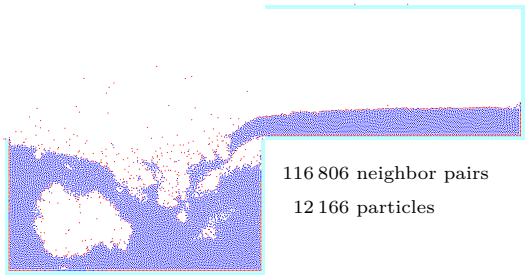
(f) Our method after 40 seconds



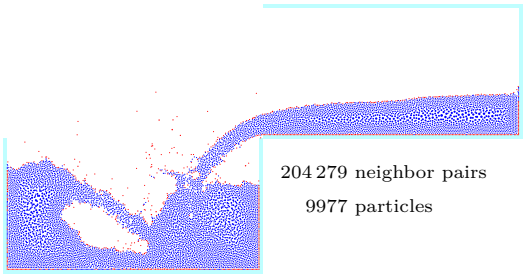
(g) Basic PBF after 45 seconds



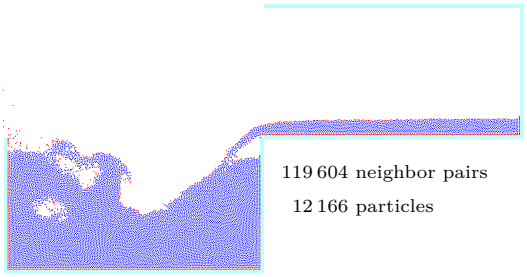
(h) Our method after 45 seconds



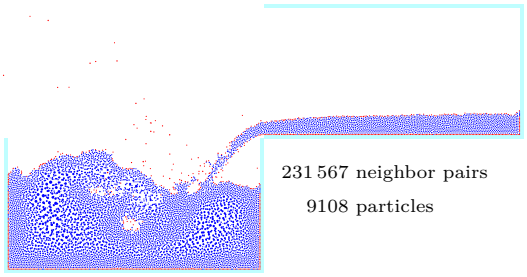
(i) Basic PBF after 50 seconds



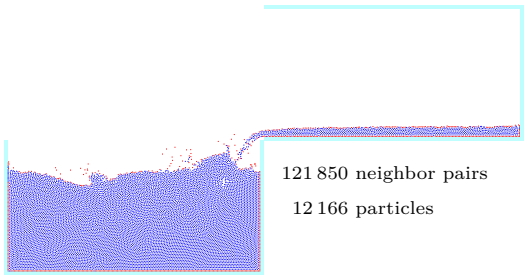
(j) Our method after 50 seconds



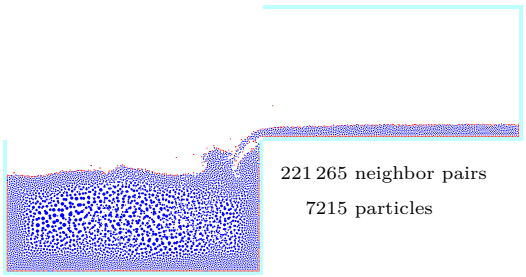
(k) Basic PBF after 60 seconds



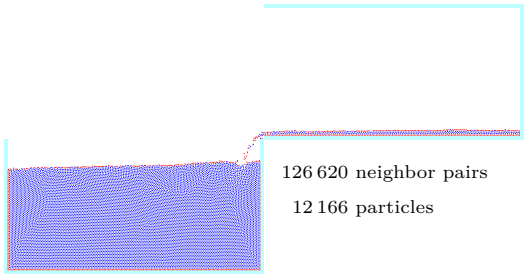
(l) Our method after 60 seconds



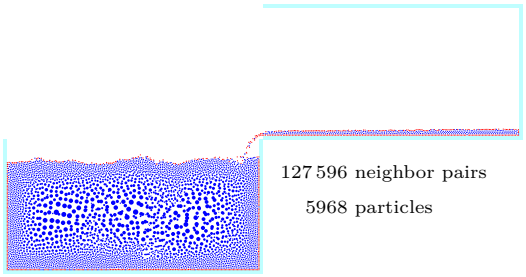
(m) Basic PBF after 70 seconds



(n) Our method after 70 seconds

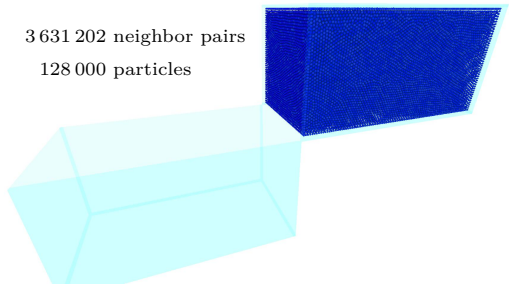


(o) Basic PBF after 90 seconds

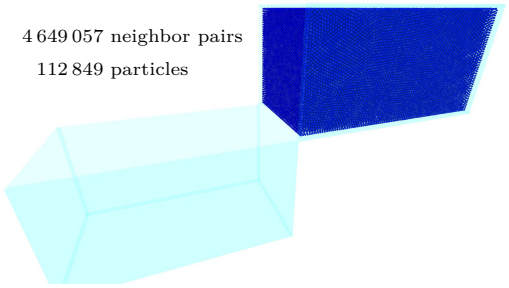


(p) Our method after 90 seconds

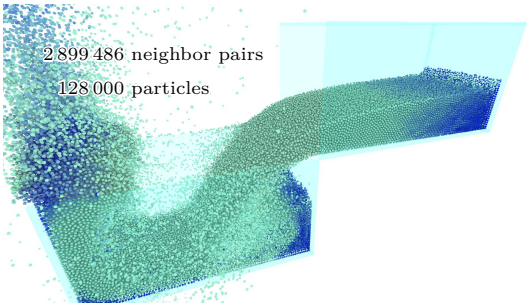
Figure 6.7: A two-dimensional test scenario containing a waterfall: The left column shows snapshots of a simulation using basic PBF, while the right column contains snapshots of a simulation using our method. The red particles are classified as boundary particles. The left wall of the upper container vanishes after 30 seconds, during which our method merged most particles in the middle of the fluid. When the amount of fluid in the upper container diminishes, the big particles are split so that the thin stream of water is simulated with the highest allowed detail. In (l), (n), and (p), the fluid calms down and the particles in the fluid center start to merge again.



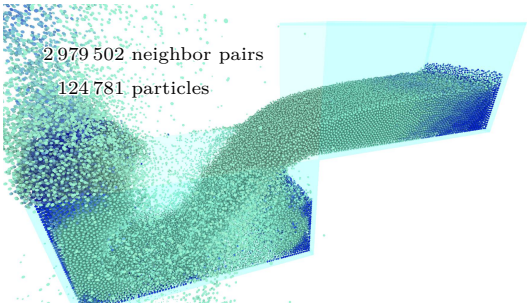
(a) Basic PBF after 30 seconds



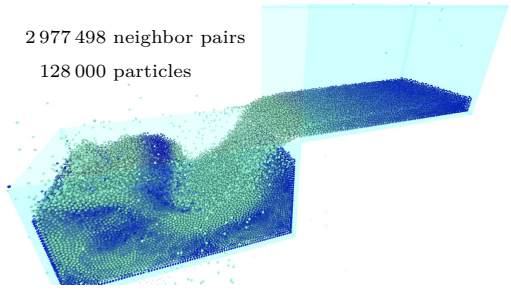
(b) Our method after 30 seconds



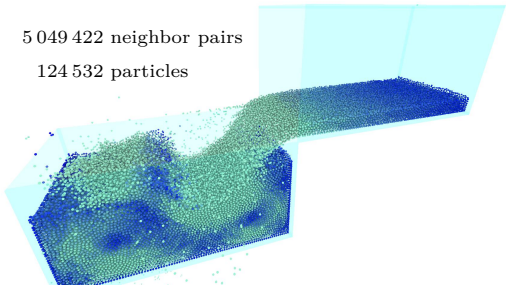
(c) Basic PBF after 43 seconds



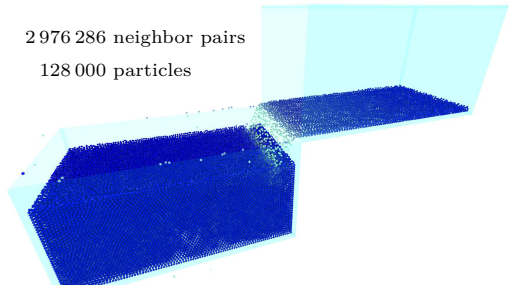
(d) Our method after 43 seconds



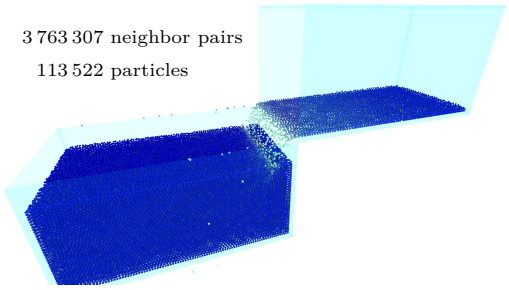
(e) Basic PBF after 58 seconds



(f) Our method after 58 seconds



(g) Basic PBF after 90 seconds



(h) Our method after 90 seconds

Figure 6.8: A test scenario similar to Figure 6.7, but in three dimensions. The particle color encodes the velocity. Both methods result in very similar particle positions and velocities.

	Simulation Step		Neighbor Search		Constraint Solver	
	Basic	Our Method	Basic	Our Method	Basic	Our Method
30 s	5.658	**4.670**	4.087	**2.084**	1.490	**1.457**
34 s	5.569	**4.830**	4.086	**2.346**	1.402	**1.343**
40 s	5.581	**4.757**	4.169	**2.623**	1.329	**1.138**
45 s	**5.558**	6.097	4.167	**3.468**	**1.309**	1.510
50 s	**5.592**	6.138	4.212	**3.271**	**1.298**	1.823
60 s	5.626	**5.403**	4.207	**2.612**	**1.336**	1.824
70 s	5.665	**5.554**	4.208	**2.413**	**1.374**	2.056
90 s	5.644	**4.579**	4.162	**2.079**	**1.399**	1.421

Table 6.5: The computation time of each snapshot listed in Figure 6.7. All measured times are given in milliseconds, measured using an NVIDIA RTX 2070. The column "Simulation Step" contains all computations necessary for the simulation, including the neighbor search, the constraint solver, the other steps of PBD, and the additional steps needed for our method (like boundary distance computation, merging/splitting, etc.). During the pouring phase, the big particles are split into smaller particles causing the particle count and the neighbor pair count to increase. The larger number of particles increases the neighbor search duration, and the larger number of neighbor pairs increases the constraint solver duration. The overhead of merging/splitting—and especially the creation and deletion of particles—occasionally causes our method to be slower than basic PBF during the pouring phase, but during phases with sufficient fluid depth, the performance gain due to the reduced particle count outweighs the overhead.

	Simulation Step		Neighbor Search		Constraint Solver	
	Basic	Our Method	Basic	Our Method	Basic	Our Method
30 s	**10.607**	18.712	**5.177**	10.014	**5.324**	6.898
43 s	**9.286**	12.241	**5.065**	6.392	**4.116**	4.285
58 s	**9.389**	20.909	**4.921**	11.688	**4.363**	7.399
90 s	**9.407**	18.137	**4.949**	10.707	**4.353**	5.788

Table 6.6: The computation time of each snapshot listed in Figure 6.8. All measured times are given in milliseconds, measured using an NVIDIA RTX 2070. These results show the drawback when using our method on fluids with insufficient depth.

CHAPTER 7

Conclusion and Future Work

In this thesis, we have presented a modification to PBF that allows fluid particles to adapt in size according to the fluid shape so that the number of particles is lowered and the computational effort can be reduced. We have analyzed how fluid particles of different sizes should interact with each other while maintaining results consistent with basic PBF. A crucial step in this regard is to prevent missing out on relevant particles in the neighborhood (especially referring to larger neighboring particles). An approach for smoothly transitioning between different fluid granularities (i.e., the particle sizes) using gradual merging and splitting has been described in detail. For particle splitting, we have discussed the properties and tradeoffs of gradual and instant splitting strategies. Furthermore, we have proposed to use the fluid depth (defined as the distance to the fluid boundary) for choosing the fluid granularity at any given location in the fluid. Using this criterion enables further simplifications, leading to a concrete streamlined implementation variant which we have described in Section 4.6. In addition, we have analyzed our method mathematically, showing that our method has the potential to increase a PBF implementation's performance through overall reduced neighborhood evaluations. Expressing the expected number of neighbor pairs in a mathematical equation has shown that for shallow fluid bodies, our method might increase the number of neighbors, but with increasing fluid depth, the neighbor count quickly falls below that of basic PBF.

There are still some problems that have to be addressed in future research: The performance of our test implementation has to be improved so that running tests with higher numbers of particles becomes feasible at real-time frame rates. Our decision to let the fluid resolution exclusively depend on the boundary distance allows to simplify our algorithm, but it also means that the fluid bodies need to be of considerable size (depth) until our method actually reduces the total number of neighbors. Exploring alternative criteria for the target radius computation seems promising for the purpose of further particle count reductions. A possible starting point might be to not set every boundary

particle immediately to the smallest target radius, but to choose the assigned target radius depending on further criteria, e.g., the fluid's local feature size as used by Adams et al. [APKG07] in their SPH simulation.

Regarding the future development of adaptive sampling in PBF, a desirable goal might be the creation of a method that allows to specify an upper limit on the number of particles, where particle merging and splitting are utilized to always obey this limit.

Another aspect worth analyzing might be the effect of the larger kernel sizes on the propagation speed of pressure throughout the fluid. In our tests, we did not notice any negative effects caused by the local variations of the kernel widths, but with increasing particle size differences, this might lead to problems that have to be addressed explicitly.

Nomenclature

Per Particle Properties (of Particle with Index i)

m_i	Particle mass
r_i	Particle radius (for implicitly storing ρ_{0i} of every fluid particle)
$\mathbf{v}_i$	Particle velocity vector
$\hat{\mathbf{v}}_i$	New particle velocity vector after constraint solving
$\mathbf{p}_i$	Particle position (during constraint solving); also used to refer to specific particles
$\hat{\mathbf{p}}_i$	New particle position after constraint solving
$\mathbf{x}_i$	Old particle position from the previous timestep
b_i	Boundary distance
$\check{r}_i$	Target radius
ρ_i	Fluid's current density at position $\mathbf{p}_i$
ρ_{0i}	Fluid's rest density (might vary between particles if they represent different fluids)

Per Constraint Properties (for Constraint with Index j)

λ_j	Lagrange multiplier used during constraint solving
l	Target distance in distance constraints
h	Kernel width
$\check{h}_j$	Intrinsic kernel width

Global Properties

n	Total number of particles
d	Dimensionality of the simulation

$\mathbf{p}$ Vector containing all particle positions (i.e., all $\mathbf{p}_i$)

Δt Predicted duration of the timestep

$r_{\min}$ Smallest allowed particle radius

Functions

$C_j(\mathbf{p})$ Constraint

$\hat{C}_j(\mathbf{p})$ Linearization of the constraint (tangential plane at a specific position)

$W(\mathbf{p}, h)$ Kernel function with kernel width h ($\mathbf{p} = \mathbf{0}$ is at kernel center)

$f_h(b)$ Function mapping boundary distance to kernel width

$f_{\check{h}}(b)$ Function mapping boundary distance to intrinsic kernel width

$f_{\check{r}}(b)$ Function mapping boundary distance to target radius

$f_h(x)$ Function mapping fluid center distance to kernel width

$f_{\check{h}}(x)$ Function mapping fluid center distance to intrinsic kernel width

$f_{\check{r}}(x)$ Function mapping fluid center distance to target radius

$f_{\mathrm{d}}(x)$ Function mapping fluid center distance to particle density

$f_{\mathrm{n}}(x)$ Function mapping fluid center distance to number of neighbors per particle

$f_{\mathrm{p}}(x)$ Function mapping fluid center distance to neighbor pair distribution

Symbols Used for Split and Merge

s Source particle (during split/merge)

t Target particle (during split/merge)

m_s Mass of the source particle (before split/merge)

m_t Mass of the target particle (before split/merge)

$\hat{m}_t$ Mass of the target particle (after split/merge)

r_s Radius of the source particle (before split/merge)

r_t Radius of the target particle (before split/merge)

$\hat{r}_t$ Radius of the target particle (after split/merge)

$\mathbf{p}_s$ Position of the source particle (before split/merge)

$\mathbf{p}_t$ Position of the target particle (before split/merge)

$\hat{\mathbf{p}}_t$	Position of the target particle (after split/merge)
$\mathbf{v}_s$	Velocity vector of the source particle (before split/merge)
$\mathbf{v}_t$	Velocity vector of the target particle (before split/merge)
$\hat{\mathbf{v}}_t$	Velocity vector of the target particle (after split/merge)
q	Transferred part during split/merge
t_{left}	Remaining duration during a split/merge
c_{m}	Factor used to perform merges
c_{s}	Factor used to perform splits

Other Symbols

i	Integer value used for indexing particles
j	Integer value used for indexing constraints or neighbor particles
$\mathbf{M}^{-1}$	Diagonal matrix containing the inverse particle masses
$\mathbf{M}^{-1/2}$	$\mathbf{M}^{-1}$ with component-wise application of the square root
$\mathbf{s}$	Sampling position for the fluid density
χ_p	Center of mass of a set of particles
a	Gradient of $f_h(b)$
g	Number of digits in a binary number
$\bar{g}$	Number of deleted digits in a binary number
z_{n}	Set of neighboring Z-Order indices
r_{f}	Radius of a spherical fluid body
x	Distance from fluid center
k	Kernel scale
p_{e}	Expected number of neighbor pairs
y_1, y_2, y_3	Integration variables
w_i	Weights during kernel sampling (kernel gathering and spreading)

Mathematical Notation

p_{ix}	The x-coordinate of the position of the particle with index i

$\Delta\mathbf{p}$ Displacement of the particle position during constraint solving

$\nabla C_j(\mathbf{p})$ Gradient of the constraint function

$\nabla_{\mathbf{p}_i} C$ Gradient of the constraint function if only the position of the particle with index i is variable

$\frac{\partial C}{\partial p_u}$ Gradient of the constraint function if only the u-th value of the position vector is variable

$|\mathbf{v}|$ Length of vector $\mathbf{v}$

(x, y) Point with the coordinates x and y

Acronyms

FLIP Fluid-Implicit-Particle. 6

PBD Position-Based Dynamics. vii, ix, 2, 3, 5, 9–12, 16, 17, 20, 47, 48, 68

PBF Position-Based Fluids. vii, ix, 3, 5, 17, 19–21, 27, 29, 32, 36, 41, 47, 48, 55–59, 61–70

SPH Smoothed Particle Hydrodynamics. vii, 3, 5–7, 17

Bibliography

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. In *ACM SIGGRAPH 2007 papers*, pages 48–es. Association for Computing Machinery, 2007.

[Ben07] Jan Bender. *Impulsbasierte Dynamiksimulation von Mehrkörpersystemen in der virtuellen Realität*. PhD thesis, University of Karlsruhe, 2007.

[BET14] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive simulation of rigid body dynamics in computer graphics. *Computer Graphics Forum*, 33(1):246–270, 2014.

[BFS05] Jan Bender, Dieter Finkenzeller, and Alfred Schmitt. An impulse-based dynamic simulation system for VR applications. In *Proceedings of Virtual Concept*, volume 2, 2005.

[BMM17] Jan Bender, Matthias Müller, and Miles Macklin. A survey on position based dynamics, 2017. In *Proceedings of the European Association for Computer Graphics: Tutorials*, EG '17, pages 1–31, Goslar, DEU, 2017. Eurographics Association.

[Bri16] Robert Bridson. *Fluid simulation for computer graphics*. CRC press, 2016.

[Dim07] Rouslan Dimitrov. Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*, 2007.

[Gre08] Simon Green. Cuda particles. *NVIDIA whitepaper*, 2(3.2):1, 2008.

[HHK08] Woosuck Hong, Donald H House, and John Keyser. Adaptive particles for incompressible fluid simulation. *The Visual Computer*, 24(7-9):535–543, 2008.

[Khr21] The Khronos Vulkan Working Group. *Vulkan 1.2.170 – A Specification*, 2021.

[KK16] Marcel Köster and Antonio Krüger. Adaptive position-based fluids: Improving performance of fluid simulations for real-time applications. *International Journal of Computer Graphics & Animation*, 6(3):01–16, 2016.

[Mac20] Miles Macklin. NVIDIA FleX, Nov 2020. `https://developer.nvidia.com/flex`, Accessed: 2020-12-07.

[MCG03] Matthias Müller, David Charypar, and Markus H Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, 2003.

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[MM13] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics*, 32(4):1–12, 2013.

[MMCK14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 33(4), July 2014.

[Mon92] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

[Mor66] Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

[Mül08] Matthias Müller. Hierarchical position based dynamics. In *Workshop in Virtual Reality Interactions and Physical Simulation*. The Eurographics Association, 2008.

[NVI18] NVIDIA Corporation. NVIDIA Turing GPU architecture. 2018.

[Rey83] Osborne Reynolds. An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels. *Philosophical Transactions*, 174:935–982, 1883.

[SG11] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. In *ACM SIGGRAPH 2011 papers*, pages 1–8. Association for Computing Machinery, 2011.

[TH81] Herbert Tropf and Helmut Herzog. Multidimensional range search in dynamically balanced trees. *Angewandte Informatik*, (2):71–77, 1981.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214, 1987.

[ZSP08] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. Adaptive sampling and rendering of fluids on the GPU. In *Proceedings Symposium on Point-Based Graphics*, pages 137–146, August 2008.