NAIST-IS-MT1151205

Master's Thesis

Adaptive Particle Splitting Based on Turbulence Energy for Fluid Simulations on GPUs

Arno in Wolde Lübke

September 20, 2013

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Master's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

Arno in Wolde Lübke

Thesis Committee:

Professor Hirokazu Kato	(Supervisor)
Professor Tsukasa Ogasawara	(Co-supervisor)
Professor Jun Miyazaki	(Tokyo Institute of Technology)
Assistant Professor Takafumi Taketomi	(Co-supervisor)

Adaptive Particle Splitting Based on Turbulence Energy for Fluid Simulations on GPUs*

Arno in Wolde Lübke

Abstract

We present a simulation method for particle-based fluids to capture small-scale features such as splashes in higher resolution than the underlying base simulation. For this purpose, we split particles in visually important regions of the fluid into smaller, high-resolution particles.

To reduce the computational overhead introduced by the additional simulation scale, we propose splitting only those particles found within turbulent surface regions that are visible to the camera. We extract these particles in screen space and decide to split them based on their turbulent energy. Further, to compensate for irregularities in the quantity field that are introduced by transitioning particles, we use a simple blending approach to maintain stability. We fully implemented our method for graphics processing units to further accelerate the computational speed.

In early experiments we could achieve speed increases of up to two over a high-resolution simulation while preserving similar visual qualities.

Keywords:

Fluid Simulation, Particle-based Fluids, Smoothed Particle Hydrodynamics, Multi-resolution Fluids, Turbulence Energy, General Purpose Computing on GPU

* Master's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1151205, September 20, 2013.

Contents

List of Figures

List of Tables

1. Introduction

Fluids such as liquids and gases are an important visual element in movies, video games and advertisements. However, to achieve visually pleasing results, simulations are often carried out at high resolution. Additionally, the underlying model of fluids and its numerical solutions requires high computational costs, which make physics-based fluid simulations impractical for real-time applications and lead to long design cycles in offline applications. To alleviate these computational requirements, this work proposes a two-scale approach for particle-based fluids and discusses its implementation on graphics processing units for further performance gains.

Recently particle-based fluids have evolved as an interesting alternative to their grid-based counterparts, since their governing equations' concept is more simple and computations are only carried out for each fluid element instead of an embedding data structure. In addition, particle-based methods trivially conserve mass.

For the purpose of solving the particle-based model, Smoothed Particle Hydrodynamics (SPH) has become a commonly used method that allows the reconstruction of continuous quantity fields by taking a weighted average of the neighboring particles' values. Using SPH, real-time results have been accomplished [23]. However, the number of particles is limited to a couple of thousands for real-time applications. In addition, small scale details such as splashes cannot be captured accurately, and the final visual result of the fluid appears to be blobby.

To simulate more particles in shorter time, graphics processing units (GPUs), which posses multiple processing units for parallel computations, have been used to speed up the computation of SPH Fluids. Speed-ups on order of ten have been reported in early results [13]. Furthermore, multi-resolution methods that simulate fluid areas at different scales depending on certain criteria, have been proposed and succeeded in preserving visual detail while significantly reducing the computational cost [30].

To our knowledge, only a few efforts have been made to combine GPUs and multi- resolution SPH fluids. Furthermore, existing implementations produced no considerable performance gains if the fluid was very turbulent [26]. This

work proposes an easy-to-implement multi-scale simulation method for graphics processing units that splits particles in visually important regions of the fluid into smaller, high resolution particles. To reduce the computational overhead introduced by the additional simulation scale, particles are only split in turbulent surface regions that are visible to the camera. For this purpose, surface particles are extracted in screen space and divided based on their turbulent energy.

The contributions of this work may be summarized as follows:

- Many implementations of particle-based fluids use screen-space rendering techniques. In this thesis one of these techniques is used to extract particles that are located on the fluid's surface. This method has the advantage that it correctly finds all surface particles visible to the camera and not mistakenly includes particles within the fluid body

- Recently, the concept of blending was introduced to multi-resolution fluids, in order to maintain stability in areas where particles are transitioning between high and low resolutions. In this work, this model is simplified to reduce the overhead that comes with the original method. In essence, in order to avoid numerical issues that come with this simplifications, this work restricts the movement of a particle based on the Courant Friedrichs Lewie condition.

- In order to further reduce the computational overhead, this thesis to proposes an additional splitting criterion based on the turbulent energy of a particle, which ensures that surface particles are only split in turbulent, and visual-important areas of the fluid.

- Since merging of particles is one of the most difficult operations to implement on GPUs and is not comprehensively described in current literature, an easy to implement merging method for GPUs is proposed.

1.1 Related Research

Algorithms for fluid animation in computer graphics rely heavily on findings from the research field of fluid mechanics. In particular the motion of a fluid can be formulated by a set of partial differential equations, the so called Navier-Stokes

Equations. These equations were formulated independently by Claude Navier in 1822 and George Stokes in 1845, respectively

Particle systems, as introduced to computer graphics by Reeves [28], are a popular model to describe natural phenomena such as liquids, smoke or clouds, that are difficult to capture using classical surface representations.

A numerical method that is commonly used in combination with a particle system to model fluid behaviour is Smoothed Particle Hydrodynamics (SPH). Though it was initially developed to describe astrophysical phenomena [8, 18], an application of SPH to free surface flows was given by Monaghan in 1994 [21].

Müller et. al. [23] demonstrated that SPH can be used in real-time fluid simulations using specially designed weighting kernels and the ideal gas equation to relate densities to pressures. This method has the disadvantage that the fluid is highly compressible. Becker and Teschner [3] popularized a weakly compressible SPH model for computer animation. Furthermore, the predictive corrective incompressible SPH (PCISPH) method [29] is a modern SPH fluid solver that allows for incompressibility at high simulation time steps. Recently, Maklin and Müller [20] presented a solver based on the Position Based Dynamics framework [24], that increases the integration time step by ten over the PCISPH method.

Bridson et. al. [4] suggested a way to create divergence free, turbulent velocity fields for procedural animation by taking the curl of a noise function. Kim et. al. [16] used this result in combination with wavelet noise [5] to add small scale detail as a post processing step. Fujisawa et. al. [7] applied these results to the SPH fluid model by formulating a wavelet decomposition for particle-based fluids and adding details using vortex sub-particles.

The goal of this work is to present methods that can accellerate these approaches to fluid simulation. Related research can generally be devided into three different approaches:

Multi-Resolution Fluids Multi-resolution approaches aim to allocate more computational resources to specially-defined fluid regions to increase the visual quality and/or physical accuracy of the simulation.

Adams et. al [1] sample the fluid using the geometric local feature size as criterion, so that complex geometric regions are represented by more particles than lesser complex areas. In addition to geometric criteria, this work

investigates the use of a physics-related property, the turbulence energy of a particle as a criterium to avoid costs that arise due to splitting and merging particles. Further this work suggests a new way to extract surface particles by exploiting the popular screen space fluid rendering algorithm of van der Laan et. al. [31]. Zhang et. al. [32] explain how adaptive sampling can be mapped to the GPU exploiting the rendering pipeline.

Orthmann and Kolb [26] present a method that allows a smooth transition between particles of different resolutions avoiding numerical problems that might occur due to poor resampling of the fluid. This work makes use of this method, but simplifies it in order reduce computational overhead and ease its implementation.

Graphics Processing Units The advent of programmable rendering pipelines in computer graphics, gave researchers the opportunity to port various algorithms to the graphics processing unit (GPU) in order to benefit from its parallel architecture. Harada et. al. [13] implemented the SPH fluid model on the GPU using OpenGL and C for Graphics. They reported that their method enabled them to use ten times as many particles as previous real-time simulations.

As programable shaders were primarily designed for computer graphics purposes, the wish to also implement algorithms of different research areas on commidity hardware led to the research area of General Purpose Computing on Graphics Processing Units (GPGPU). Application Programming Interfaces (APIs) such as NVIDIA's Compute Unified Device Architecture (CUDA) [25], allow the implementation of fluid dynamics algorithms more conveniently without the restrictions of shader programs.

NVIDIA's white paper on particles [11] showed an efficient way to compute a particle's neighborhood on GPU's and is commonly used in GPU implementations of SPH fluids. Recently, Goswami et. al. [10] made use of z-ordering to arrange particles for neighborhood queries and access particle data through fast shared memory.

This work follows the approach of [11] and shows a way to implement the splitting and merging operations of multi-resolution fluids.

Approximation In order to save computational time, approximation strategies try to avoid computations for particles from which contributions to the whole system are neglectable.

Goswami and Pajarola [9] classify fluid particles as either active or inactive. Particles are considered inactive if they do not contribute to the visual quality and their speed is below a certain threshold. Inactive particles are then not considered in the SPH computations.

Pelfrey and House [27] optimize the standard SPH approach, by computing the neighborhood of a particle only when necessary.

1.2 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 discusses the basic theory behind particle-based fluid simulations and their implemenation. It introduces the governing equations and their discretization using a particle system and the Smoothed Particle Hydrodynamics method. Further the problem of boundary handling is discussed shortly. The method developed in this chapter shall be referred as to the basic method and is built upon in the following chapter.

Chapter 3 introduces the two-scale extension made for the multi-resolution approach of this work, before it discusses a numerical robust way to dynamically handle transitions between two resolutions (*particle resampling*) using particle blending. Then resampling criteria are given by discussing the turbulence energy of a particle and its position with respect to the fluids surface. Finally, implementation details using the GPU are presented.

Chapter 4 compares the presented method with the standard SPH algorithm in a standard dambreak scenario.

Chapter 5 conludes this work and gives ideas for future work.

2. Particle-Based Fluid Simulation

This section presents a method that uses particles for simulating fluids. This method serves as the basis for further discussions in the thesis. For this purpose, first, a mathematical model for particle-based (Lagrangian) fluids is given. Next, a solution to this model is discussed by using the Smoothed Particle Hydrodynamics scheme. Finally, the special case where the fluid interacts with boundaries is briefly addressed.

2.1 Lagrangian Fluid Dynamics

Fluids, in particular liquids and gases, are substances that cannot resist shear stress, and hence continuously deform. Metaphorically speaking, under the influence of external forces, fluids take the form of their containing canvas.

The underlying physics of fluids are complicated and very hard to compute. In contrast, methods for physics-based fluid animation for computer graphics aim to simplify the model of fluids for the sake of performance, since their goal is to get visually-plausible results of fluids rather than being strictly physically accurate.

This section shall give a brief review of such a model by introducing the concepts of Lagragian fluid dynamics.

2.1.1 The Governing Equations

To simplify the the equations to be solved, in this thesis, the following assumptions are made for a fluid. A fluid is assumed to be

incompressible: the fluid is assumed to conserve its volume throughout the simulation.

isothermal: the temperature throughout the fluid body is constant.

Newtonian: the stress acting an a fluid element is proportional the the rate of strain.

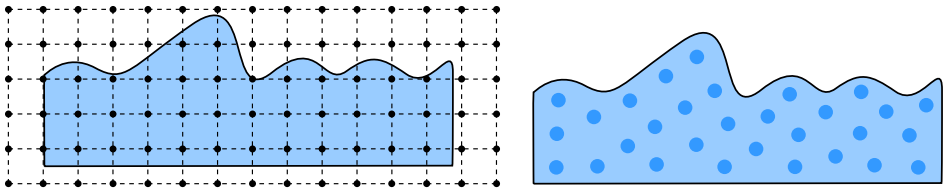


Figure 1. Comparision between the Eulerian and the Lagrangian Viewpoint: In the Eulerian viewpoint (a) fluid quantities are evaluated at fixed, uniformly distributed positions in space, while in the Lagrangian viewpoint (b) fluid quantities are observed at positions of particles that are allowed to move freely in space.

The resulting governing equations of the fluid flow are called *incompressible Navier-Stokes equations* and given by

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \mu\Delta\mathbf{v} + \rho\mathbf{g} \quad (1)$$

$$\nabla\cdot\mathbf{v} = 0 \quad (2)$$

Equation 1 is called the *momentum equation* which describes change in momentum of an infinitely small fluid element. The quantities $\rho, \mathbf{v}$ and p denote the fluid density, velocity and pressure, respectively. The term $\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}$ on the left-hand side of the equation is known as the *material derivative* and describes the rate of change of velocity at a fixed point in space. The three terms on the right side reflect the forces that act on the fluid, i.e. the forces due to pressure and viscosity as well as external forces such as gravity. *Viscosity* describes the resistance of the fluid to deform, e.g. water has low viscosity while honey as high viscosity.

Equation 2 is known as the *incompressibility condition*. This constraint preserves the fluids volume and implies constant density throughout the fluid body.

2.1.2 The Eulerian and Lagrangian Viewpoint of Fluid Motion

The governing equations discussed above describe the *Eulerian viewpoint* of fluid flow, in which the change of fluid quantities is observed at fixed points in space.

Consequently, a suitable data structure for representing a fluid on a computer is a grid (as shown in Figure 1).

In this thesis a fluid is represented as a particle system, which means fluid quantities are observed for a fluid element (particle) as it moves in time and space. The related description of fluid motion is called the *Lagrangian viewpoint*. The difference between Eulerian and Lagrangian fluid representations is illustrated in Figure 1.

In the Lagrangian fluid model the governing equations simplify greatly: Since it is common to assume constant mass for each particle, enforcing mass conservation for the fluid is trivial. Furthermore, since constant density is assumed, Equation 2 can be omitted from further discussion. In addition, since fluid quantities are observed for each particle the momentum equation reduces to

$$\rho\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\nabla p + \mu\Delta\mathbf{v} + \rho\mathbf{g} \tag{3}$$

2.1.3 Motion of a Fluid Particle

Equation 3 given, the acceleration of a particle can be computed by dividing both sides by ρ, that is

$$\mathbf{a} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{1}{\rho}(-\nabla p + \mu\Delta\mathbf{v} + \rho\mathbf{g}) \tag{4}$$

Finally, given the the equations of motion for a particle,

$$\mathbf{a} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} \qquad \mathbf{v} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \tag{5}$$

and the initial positions and velocities for all particles, one can determine the positions of all particles at any arbitratry point in time.

2.2 Smoothed Particle Hydrodynamics

Solving the governing equations of fluids on a computer requires the simulation domain to be limited to a finite set of discrete sample points.

As mentioned above, contrary to Euclidean fluid dynamics, implementations of Lagrangian fluid simulations sample the fluid body at scattered points and use a particle system to store fluid quantities such as velocity or pressure. Consequently,

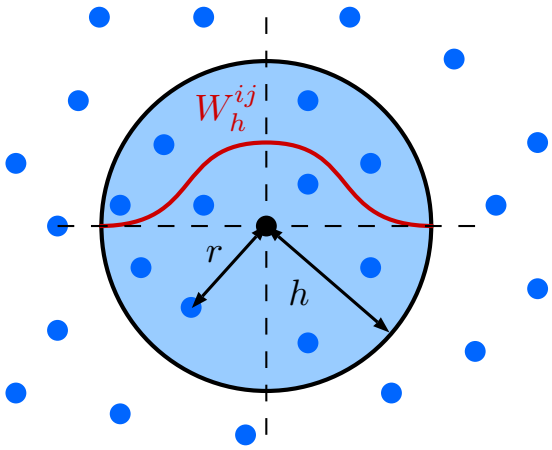


Figure 2. Reconstruction of continuous functions using SPH: For any point $\mathbf{x}$ in the simulation domain its corresponding physical quantity is calculated by taking a weighted sum of all particle quantities within the support radius h.

as time progresses, these quantities are examined at the current location of each particle.

In order to evaluate Equation 3 differential operators need to be evaluated at these positions. For this purpose numerical schemes usually reconstruct continuous functions from sampled data. A prominent choice of such a method for computer graphics is *Smoothed Particle Hydrodynamics (SPH)* .

Though initially employed for simulating astrophysical phenomena [18, 8], recent publications such as [23] promoted Smoothed Particle Hydrodynamics for the use in fluid simulations, and provided a foundation for more sophisticated methods in the field.

The rest of this section shall briefly present the SPH concepts relevant to this work. For a more thorough introduction to this topic the interested reader is referred to [2].

2.2.1 Approximating functions with SPH

Given a finite set of particles with positions $\mathbf{x}_i$ and and an arbitrary physical quantity ϕ_i, which value is known at each of these positions, $\phi(\mathbf{x})$ at any point

in space can be approximated as the sum of the weighted contributions of neighboring particles (that is, particles, whose distance to $\mathbf{x}$ is smaller or equal to a user-defined threshhold h), i.e.

$$\phi(\mathbf{x}) = \sum_j \phi_j \frac{m_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h), \tag{6}$$

where j refers to the index of each particle in the neighborhood of $\mathbf{x}$. ϕ_j is the physical quantity, m_j is the mass, ρ_j is the density of the particle with index j. See also Figure 2.

W is a so-called *kernel function* that assigns a weight to each particle within a radius $r = \|\mathbf{x} - \mathbf{x}_j\| \leq h$, where h is also known as the *effective radius* of W. Depending on the situation, different kernels might be used to weight each particle's contribution, the kernels used in this work are discussed later on in Appendix A.

Equation 6 itself may be derived from the following identity: $\phi(\mathbf{x}) = \int_{\mathbf{x}'} \phi(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')\mathrm{d}\mathbf{x}'$, where the delta distribution δ is approximated with the kernel function, and the differential $\mathrm{d}\mathbf{x}'$ is replaced by the volume $V_j = \frac{m_j}{\rho_j}$ that particle j is occupying.

As shown in Appendix A derivatives for each kernel can be computed easily, and since the sample values ϕ_j are constants, differential operators can directly be applied to the SPH approximation of a quantity field. For example the gradient of a quantity field may be approximated by

$$\nabla\phi(\mathbf{x}) = \sum_j \rho_j \frac{m_j}{\rho_j} \nabla W(\mathbf{x} - \mathbf{x}_j, h) \tag{7}$$

whereas the Laplacian of a quantity field can be computed as follows

$$\Delta\phi(\mathbf{x}) = \sum_j \rho_j \frac{m_j}{\rho_j} \Delta W(\mathbf{x} - \mathbf{x}_j, h) \tag{8}$$

Unfortunately, these expressions are not suitable in the context of fluid simulations. This is mainly due to the fact that these equations lead asymmetrical forces being exchanged between particle pairs, which in turn violates Newtons third law and prevents the particle system to conserve linear and angular momentum. In order to circumvent this problem, properties of the SPH approximation can be

exploited when deriving these operators. Multiple derivations exist. This work uses

$$\nabla\phi(\mathbf{x}) = \rho(\mathbf{x}) \sum_j \left(\frac{\phi(\mathbf{x})}{\rho(\mathbf{x})^2} + \frac{\phi_j}{\rho_j^2} \right) m_j \nabla W(\mathbf{x} - \mathbf{x}_j, h) \tag{9}$$

for evaluating the gradient, and

$$\Delta\phi(\mathbf{x}) = \sum_j (\phi_j - \phi(\mathbf{x})) \frac{m_j}{\rho_j} \Delta W(\mathbf{x} - \mathbf{x}_j, h) \tag{10}$$

for the Laplacian of a quantity field. For a more thorough discussion on equations 9 and 10 the reader is referred to [15, 2].

2.2.2 Application to Fluid Dynamics

In implementations, SPH is particularly utilized to discretize the *internal forces* acting on each particle, that is the forces due to pressure and viscosity in equation 1:

$$\mathbf{f}_i^{\text{internal}} = \mathbf{f}_i^{\text{p}} + \mathbf{f}_i^{\mu} = -\nabla p + \mu \Delta \mathbf{v} \tag{11}$$

where $\mathbf{f}_i^{\text{internal}} = \sum_j \mathbf{f}_{ij}$ is the net force that all particles in the system exert on particle i. As physical systems conserve momentum in the absence of external forces (i.e., in case of the linear momentum, $\sum_i \mathbf{f}_i^{\text{internal}} = 0$), Newton's third law, that states that forces a particle pair i, j cancel each other out (i.e. $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$) must hold. For this purpose, as discussed above, the pressure force acting on a particle i is computed, as follows:

$$\mathbf{f}_i^{\text{p}} = -\rho_i \sum_i \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{12}$$

whereas the viscosity force is given by:

$$\mathbf{f}_i^{\mu} = \mu \sum_j (\mathbf{v}_j - \mathbf{v}_i) \frac{m_j}{\rho_j} \Delta W(\mathbf{x}_i - \mathbf{x}_j, h). \tag{13}$$

Usually, different kernels for each equation are used. This work uses the kernels proposed by Müller et. al. [23], which are listed in Appendix A.

In order to evaluate pressures at particle locations, as needed by equation 12, Tait's equation

$$p_i = B\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right) \tag{14}$$

is commonly used to relate a particle's density to its pressure. B is known as Tait's constant and ρ_0 is the rest density of the fluid. In this thesis γ is chosen to be one.

The density of a fluid can be directly estimated by substituting ϕ with ρ in Equation 6, which yields,

$$\rho_i = \sum_j m_j W(\mathbf{x}_i - \mathbf{x}_j, h) \tag{15}$$

where m_j is normally assumed to be constant, in which case the equation simplifies to $\rho_i = m \sum_j W(\mathbf{x}_i - \mathbf{x}_j, h)$.

2.3 Boundary Handling

This section deals with effects that appear at the fluid boundaries. In this thesis, boundaries in particular are the areas of transition from fluid to solid body or air. Specifically, forces exerted on a particle that are due to surface tension and solid boundaries are discussed.

2.3.1 Surface Tension

Surface tension is a small-scale liquid phenomena that is the result of molecular cohesion forces. From a mathematical viewpoint surface tension can be interpreted as the force that aims to minimize a fluid's curvature.

Literature provides many surface tension models, due to its simplicity this work adopts the approach of Becker and Teschner [3], which models a particle's acceleration due to surface tension as follows

$$\mathbf{a}_i^{\sigma} = -\kappa \sum_j (\mathbf{x}_i - \mathbf{x}_j) m_j W(\mathbf{x}_i - \mathbf{x}_j) \tag{16}$$

2.3.2 Solid Obstacles

For solid obstacles, this work employs the force-based approach due to Monaghan [22]. Special boudary particles are inserted at the bounderies of each solid obstacle. The acceleration a particle i experiences due to a boundary particle k is then modelled as follows:

$$\mathbf{a}_i^b = \sum_k \frac{m_k}{m_i + m_k} \Gamma(\mathbf{x}_i, \mathbf{x}_k) \frac{\mathbf{x}_i - \mathbf{x}_k}{\|\mathbf{x}_i - \mathbf{x}_k\|}, \tag{17}$$

where Γ is given by

$$\Gamma(\mathbf{x}_i, \mathbf{x}_k) = \begin{cases} \frac{2}{3} & 0 < q < \frac{2}{3} \\ (2q - \frac{2}{3}q^2) & \frac{2}{3} < q < 1 \\ \frac{1}{2}(2-q)^2 & 1 < q < 2 \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

and $q = 2\frac{\|\mathbf{x}_i - \mathbf{x}_k\|}{h}$.

2.4 Summary

The goal of this section was to provide the reader with the implementation details for the particle-based fluid simulations which this thesis builds upon.

In summary, a simulation step can be subdivided into four parts. In the first part, for each particle all neighbors (that is, particles within the effective radius of a particle) are found (or rearranged in a way so that they can easily be found throughout the SPH computations). A popular method for CPU implementations is spatial hashing as described in [19] . A common approach on GPUs is to sort particle according their positions in a uniform grid that samples the simulation domain. In the second part, the density and pressure at each particle's position are computed as described in Section 2.2.2. In the third part the net force acting on each particle is determined as discussed in Sections 2.2.2 and 2.3. Finally, the system is integrated in time. Several time integration schemes such as the leapfrog or the Verlet method can be used. However, for the purpose of this thesis, simple Euler integration has been found to be sufficient.

The procedure of this method is illustrated in the following algorithm:

while *animating* **do**
  foreach *particle i* **do**
    compute particle neighbors
  end
  foreach *particle i* **do**
    compute density ρ_i *(Eq. (15))*
    compute pressure p_i *(Eq. (14))*
  end
  foreach *particle i* **do**
    compute acceleration $\mathbf{a}_i \leftarrow \frac{1}{\rho_i}(\mathbf{f}_i^p + \mathbf{f}_i^\mu) + \mathbf{a}_i^\sigma + \mathbf{a}_i^b + \mathbf{g}$ *(Eqns. (12, 13, 16, 17))*
  end
  foreach *particle i* **do**
    `integrate(`$\mathbf{x}_i$`, `$\mathbf{v}_i$`, `$\mathbf{a}_i$`);`
  end
endwhile

In the following section this method is extended so that it supports the dynamic sampling of a fluid with particles of different scale, so that computational resources can be saved in areas of the fluid that are of little interest. Furthermore, a criteria to identify these areas that is based on the turbulence energy of the particles is discussed.

3. Turbulence-Based Adaptive Particle Splitting

The following subsections present the proposed multi-resolution method of this work. The first subsection extends the basic method of the previous section so that particles of two different scales, that is high- and low-resolution particles, can interact with each other in a physically-plausible way. The second subsection presents a way to resample the fluid, i.e. replacing low- resolution particles with high-resolution particles or vice-versa. The third subsection explains about the criteria for resampling used in this thesis. At last, implementation apects for GPUs are exhibited.

3.1 Two-Scale Extension

The following is a short discussion of the changes to be made to the basic model, described in the previous chapter, as required by two-scale particle simulations.

As described earlier, the goal of using particles at different resolutions, is to allocate additional resources in interesting areas of the fluid body. Following work of Solenthaler et. al.[30], this work restricts itself to using only two different resolutions. The benefits of this approach are fewer errors due to numerical approximations, less implementation costs and the observation that it is visually-sufficient to describe fluid regions either with *high-resolution particles* or *low-resolution particles*.

3.1.1 Preliminaries

For the same volume, a low-resolution particle system of n_L particles has a high-resolution counter part of $n_H = k^d n_L$ particles, where $d = 2, 3$ is the dimension and $k = 2^i, i = 1, 2, 3...$ denotes the increase in resolution. In this work, the resolution of the higher-resolution particle system is usually twice the resolution of the low-resolution particle system (i.e. $k = 2$), e.g. a low resolution particle may be replaced by eight high resolution particles, as this is a common choice in literature [30].

All following discussions of this work restrict themselves to the three dimensional case where the amount of the high-resolution particles is eight times the amount of the low-resolution particles.

3.1.2 Additional Simulation Parameters

Introducing a fluid of different resolution to the simulation requires additional parameters. In particular, these additional parameters are the effective radius and the mass for the new resolution. In the following, the effective radius for the lower resolution is denoted by h_L while the effective radius for the higher resolution is denoted by h_H, and the mass of a lower resolution particle is denoted by m_L while the mass of the higher resolution particle is denoted by m_H.

The high and low resolution parameters relate to each other as follows,

$$h_L = kh_H \tag{19}$$

$$m_L = k^d m_H \tag{20}$$

i.e. for doubled resolution and for three dimensions the effective radius of a low resolution particle is twice the effective radius of a high resolution particle[1] , and the mass of a low resolution particle is eight times the mass of a high resolution particle. Since two effective radii are used, defining a particle's neighborhood is not as straightforward as in the basic method, as otherwise Newton's third law would be violated. This is due to the possibility that a low resolution particle may exert a force on a high resolution particle but not vice versa, as the low resolution particle may lay outside the effective radius of the high resolution particle as shown in Figure 3. It is therefore necessary to include this low resolution particle in the set of neighbors of the high resolution particle.

To resolve this issue, as suggested in [6], i and j may be regarded as neighbors, if

$$\|\mathbf{x}_i - \mathbf{x}_j\| \leq \max(h_i, h_j) \tag{21}$$

[1] This can be illustrated as follows: As described in Appendix B the effective radius of a particle can be determined using the following equation: $h = \left(\frac{3}{4\pi}\frac{\overline{n}}{n}V\right)^{1/3}$, where $\overline{n}$ is the (user-defined) average amount of particles within the effective radius which should be the same for both resolutions, n is the total amount of particles and V is the initial volume of the fluid. Given the effective radius of the low-resolution particles $h_L = \left(\frac{1}{n_L}\right)^{1/3}\left(\frac{3}{4\pi}\overline{n}V\right)^{1/3}$ the effective radius of the high-resolution particles relates to h_L as follows:
$h_H = \left(\frac{1}{n_H}\right)^{1/3}\left(\frac{3}{4\pi}\overline{n}V\right)^{1/3} = \left(\frac{1}{8n_L}\right)^{1/3}\left(\frac{3}{4\pi}\overline{n}V\right)^{1/3} = \frac{1}{2}\left(\frac{1}{n_L}\right)^{1/3}\left(\frac{3}{4\pi}\overline{n}V\right)^{1/3} = \frac{1}{2}h_L$

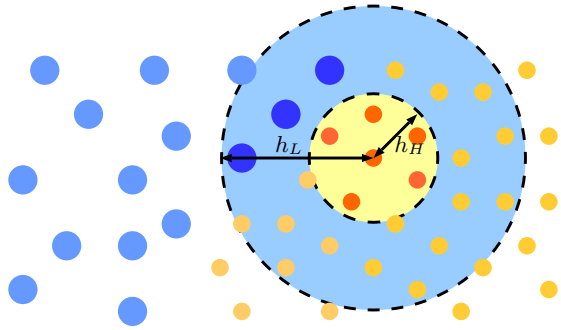


Figure 3. Definition of neighboring particles: Aside from the high resolution particles within h_H, low-resolution particles within h_L should be regarded as neighbors of particle i, in order to obey Newton's third law.

where h_i and h_j are the effective radii of particle i and j.

The definition of neighboring particles is illustrated in Figure 3.

3.1.3 Changes to SPH Terms

Due to the adjusted neighbor definition and to ensure Newton's third law, one has to ensure that particles of different resolution exert the same force on each other. In practice [6, 1], a common method to achieve this for two particles i and j, is to take the arithmetic mean of their smoothing kernels evaluated with their distance and respective effective radius (i.e. $(\nabla W(\mathbf{x}_i - \mathbf{x}_j, h_i) + \nabla W(\mathbf{x}_i - \mathbf{x}_j, h_j))/2$) to weight specific quantities.

For instance, the SPH term to compute the pressure force acting on a particle i is rewritten as

$$\mathbf{f}_i^p = -\rho_i \sum_{j \neq i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \frac{\nabla W_{ij}^{h_i} + \nabla W_{ij}^{h_j}}{2}. \tag{22}$$

The SPH terms for density, viscosity force and surface tension are adjusted similarly.

3.2 Particle Resampling

Particle resampling is needed to dynamically change resolutions in certain areas of the fluid depending on certain criteria. E.g. it might be beneficial to replace a low-resolution particle with high-resolution particles as it approaches the fluid surface. On the other hand, it would be desirable for non-visible high-resolution particles to merge into low-resolution particles in order to save computational time. Replacing particles, however, might lead to poor positioning of the newly inserted particles which results in high density fluctuations within the fluid. In these situations the arising high pressure forces require small simulation time steps to ensure stability.

To maintain large time steps, recently Orthmann and Kolb [26] introduced the concept of blending to the field of particle-based fluids. This work investigates an alternative to this method, by limiting a particles motion based on the Courant Friedrichs Lewy (CFL) condition, in order to omit parts of the computational overhead introduced by the original method, as described in the following.

3.2.1 Particle Blending

In computer graphics the term blending refers to the overlapping of two transparent layers. The result of this process is a mixed color that depends on the transparency of both layers. Decreasing the transparency of one layer while simultaneously increasing the transparency of the other layer results in a smooth transition from the color of one layer to the color of the other layer. Following this concept, instead of directly replacing particles of one resolution with particles of the other resolution, it is recommended to let both particles' resolutions coexist for a certain time period. During this time period the contribution to the system of newly inserted particles is steadily increased whereas the contribution of the particles that were supposed to be replaced is decreased. Finally, if the contribution of a particle is zero, it is removed from the system. This way, the potential high pressure forces introduced by the new particles are attenuated over a certain period of time allowing the system to reach to a stable configuration before particles are finally being replaced. The process of particle blending is illustrated in Figure 4.

For implementing particle blending each particle is assigned a state s. A par-

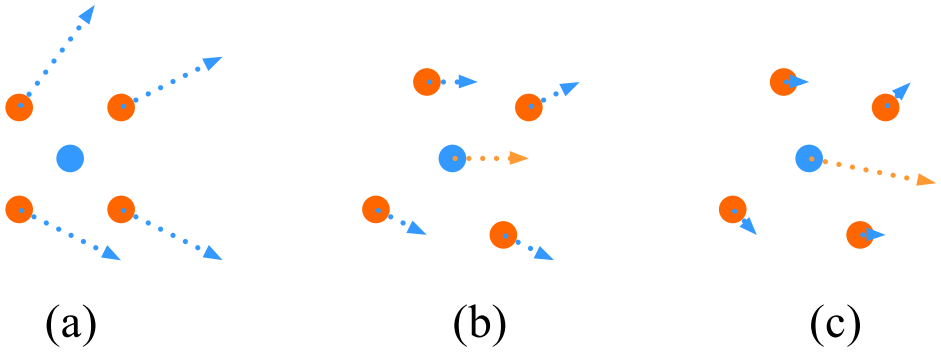


Figure 4. Particle Blending: (a) shows the situation when four new high-resolution particles (orange) where just inserted. All particles have a blending coefficient of zero and, therefore, do not contribute to the system. The low resolution (blue arrow), that is about to be deleted, has still a blending coefficient of one and therefore fully contributes to the system (blue arrows), that is it exerts an unattenuated force on the surrounding particles (that also includes the newly inserted high resolution particles). (b) shows the situation after a short period of time has elapsed: The low-resolution particle's force is attenuated, whereas the high-resolution particles' contribution (blue arrows) has increased. (c) shows the situation shortly before the low-resolution particle is being removed. Its exerted force is strongly attenuated, whereas the high-resolution particles' contribution to the system is almost one hundred percent.

ticle is in `insert` state if it is about to be inserted to the system, in `delete` state if it is about to be replaced or in `default` state if it is neither being inserted nor being replaced. Further, each particle has a blending coefficient Λ that determines its contribution to the system. Each time step, the blending coefficient of a particle i is updated depending on its state in the following way:

$$\Lambda_i^{n+1} = \Lambda_i^n + \begin{cases} \Delta\Lambda & \text{if} s_i = \texttt{insert} \\ -\Delta\Lambda & \text{if} s_i = \texttt{delete} \\ 0 & \text{if} s_i = \texttt{default} \end{cases} \tag{23}$$

where $\Delta\Lambda$ is a constant blending increment. For a newly inserted particle the blending coefficient is set to zero (meaning it has no contribution to the system)

and increased to one (meaning it contributes fully to the system) using the equation above. Likewise the blend coefficient for a particle that is supposed to be replaced starts at one and is decreased to zero, before it is finally removed from the system.

Eventually, the blending coefficient is used to weight the force that a particle exerts on another particle. For instance, the pressure force acting on particle i may now be computed by

$$\mathbf{f}_i^p = -\rho_i \sum_{j \neq i} \Lambda_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \frac{\nabla W_{ij}^{h_i} + \nabla W_{ij}^{h_j}}{2}. \tag{24}$$

taking into account the contributions of the neighboring particles j to the system.

Unfortunately, even with the use of particle blending in some cases the system may experience instabilities due to inaccurate sampling. In order to make the simulation more robust, this work restricts the motion of a particle according to the CFL condition which is a necessary condition for stability. The CFL condition basically states that per time step Δt information should not be propagated further than the spatial increment Δx of the underlying grid, that is, in one dimension, for a uniform grid $\frac{u \Delta t}{\Delta x} \leq 1$, where u denotes the maximum velocity throughout the domain, must hold. Translating this condition to a particle-based discretization of the simulation domain, this work sets the spatial increment Δx proportional to the effective radius of the simulation h, which results in the following constraint for the velocity of a particles

$$\|\mathbf{v}\| \leq \frac{\alpha h}{\Delta t} \tag{25}$$

with some constant α. Consequently in the integration step of the proposed method the computed velocity $\mathbf{v}_i$ for a particle i is adjusted as follows

$$\mathbf{v}_i^* = \min \left(\|\mathbf{v}_i\|, \alpha \frac{h}{\Delta t} \right) \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \tag{26}$$

in order to ensure this constraint given by Equation 25.

3.2.2 Particle Splitting

When a low resolution particle i is determined to be split, in three dimensions, eight new high resolution particles are inserted to the system. These particles

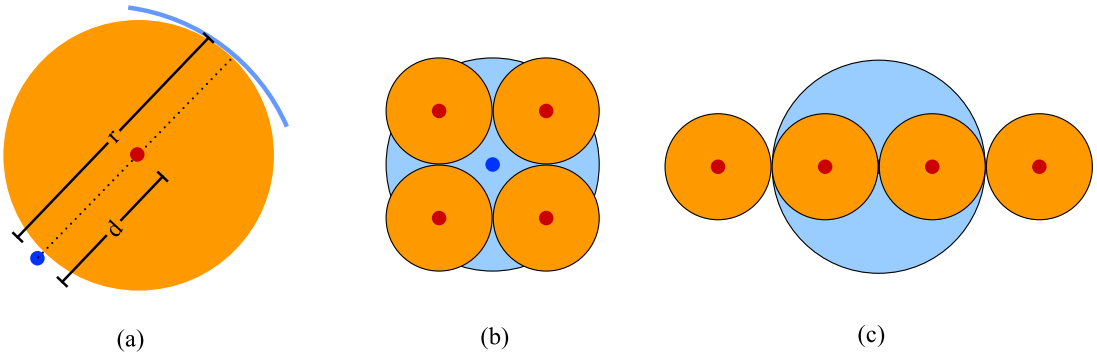


Figure 5. Splitting and merging: (a) shows the initial distance of a high-resolution particle to its parent low-resolution particle (here $d \approx \frac{1}{2}r_i$). (b) shows how high-resolution particles are aligned when being inserted. It also shows a valid configuration for high-resolution particles to merge into a single low-resolution particle. (c) shows an undesired configuration for high-resolution particles to merge.

are positioned on an axis-aligned cube around the low resolution particle, as illustrated by Figure 5(b). The distance of each high resolution particle to the low resolution particle should be proportional to the volume the low resolution particle occupies (see Figure 5(a)), and is computed as follows,

$$d \propto \frac{1}{2}r_i = \frac{1}{2}\left(\frac{3V_i}{4\pi}\right)^{\frac{1}{3}} = \frac{1}{2}\left(\frac{3m_L}{4\pi\rho_i}\right)^{\frac{1}{3}} \tag{27}$$

The velocity of all newly inserted particles equals the velocity of the low resolution particle. Finally, the low resolution particle's state is set to `delete` and the states of the high-resolution particles are set to `insert`.

3.2.3 Particle Merging

Merging saves computational time in regions of the fluid that do not require high spatial resolutions. If, in three dimensions, eight high-resolution particles, which are close to each other, are part of these regions, they should be combined combined to one low resolution particle. Position and velocity of this particle are

computed as the average over the high resolution particles that are being merged. That is, a newly inserted low-resolution particle i, has the following position and velocity:

$$\mathbf{x}_i = \frac{1}{8} \sum_{j=1}^{8} \mathbf{x}_j \tag{28}$$

$$\mathbf{v}_i = \frac{1}{8} \sum_{j=1}^{8} \mathbf{v}_j \tag{29}$$

where $\mathbf{x}_j$ and $\mathbf{v}_j$ with $j = 1..8$ are the positions and velocities of the low resolution particles to be merged.

It should be noted that, in addition to being close to each other, the particles to be merged should be aligned in a way so that they fill out the volume of the to be inserted low-resolution particle as shown in Figure 5(b) + (c). This constraint is important to improve the sampling quality. Especially in GPU implementations, identifying high-resolution particles that can be merged is not an easy task. Section 3.4.3 of this work proposes an implementation that easily can be mapped to parallel architectures.

3.3 Resampling Criteria

This subsection proposes the criteria used in this thesis to judge where the fluid particles should be split or merged. It is rather common to split particles according to geometric criteria especially in regions on and near the fluids surface [1, 30, 10]. However due to the overhead of the two-scale particle model, current multi-resolution implementations report only little to none speed increases on GPUs when the fluid is very turbulent [26]. For this purpose, this subsection proposes an additional splitting criterion that depends on the fluids turbulence energy to restrict the resampling of the fluid only to regions that exceed a certain turbulence energy threshold. Furthermore, a new method to extract surface particles that is based on screen space rendering is proposed to exactly identify only surface particles that lie within the view frustum of the camera.

3.3.1 Turbulence Energy

To identify turbulent regions within the fluid this thesis uses a wavelet-based approach similar to [16]. Wavelets are a mathematical tool to hierarchically decompose a function into a lower resolution part of the function itself and the details that are beeing lost due to the decrease in resolution. The latter part being known as *detail coefficients*. Applied to the velocity field of a fluid, the detail coefficients may be seen as a measure of the local differences of a certain velocity component and are therefore a suitable criterion for turbulence. Unfortunately, the discrete wavelet decomposition can only be applied to grids. For this reason particle data needs to be projected onto grid cells, this, however, would cause additional numerical diffusion and impose more strict memory and computational requirements. To avoid these issues this thesis follows the approach of Fujisawa et. al. [7] who define a discrete wavelet transform for particles as follows,

$$\hat{u}_i = \frac{1}{\sqrt{s}\psi_{sum}} \sum_j u_j \psi((\mathbf{x}_i - \mathbf{x}_j)/s) \tag{30}$$

where $\hat{u}_i$ is the wavelet transform of the first velocity component of particle i, ψ is the Mexican hat mother wavelet, s is the wavelet scale and $\psi_{sum} = \sum_j \psi_{sum}((\mathbf{x}_i - \mathbf{x}_j)/s)$ is a norm to address undesirable effects at the boundary of the fluid. Given the wavelet transform of the velocity $\hat{\mathbf{v}}_i = (\hat{u}_i, \hat{v}_i, \hat{w}_i)$ of a particle i, its turbulent energy is then defined as

$$E_i = \frac{1}{2}\|\hat{\mathbf{v}}_i\|^2. \tag{31}$$

Given the energy distribution of the fluid, a low-resolution particle is then determined to be split if its energy exceeds a certain threshhold E^θ. In the same way a high-resolution particle is marked for merging if its energy is below E^θ.

3.3.2 Surface Areas

As only surface areas of the fluid are directly visible to the camera, they are a desirable region of the fluid to be sampled at higher resolutions. To extract this area in previous research commonly the distance of a particle to the center of mass of its neighbors is computed [30]. If this distance is higher than a user-defined

threshold the particle is recognized to be a surface particle. That is, a particle i is a surface particle, if

$$\|\mathbf{x}_{i,cm}\| \geq d^{\theta}, \tag{32}$$

where d^{θ} is the user-defined threshold and $\mathbf{x}_{i,cm} = \mathbf{x}_i - \sum_j \mathbf{x}_j / \sum_j 1$ is a vector that points from the center of mass of the particle to the particle itself.

However, this method does not always select surface particles correctly, especially if the fluid is compressible, which may be desirable in order to achieve higher integration time steps. Moreover, this method also detects, surface particles that might not directly be visible to the camera, e.g. particles at the back of a fluid or particles outside the view frustum of the camera. For this purpose, this thesis proposes to extract surface particles in screen space, while rendering the fluid as described further in section 3.5.2.

3.4 Implementation

As discussed in subsection 2.4 the basic SPH algorithm can be separated into the three distinct operations of

- computing the density of all fluid particles
- computing the force acting of all fluid particles
- computing the new position of all fluid particles

Each operation performs (mostly) the same tasks on different data (that is, quantities of different particles) and is therefore suitable to be implemented in parallel for each particle to achieve greater performance.

As graphics cards are widely available and comparably cheap in prize, they are a desirable platform for parallel SPH implementations.

This subsection explains how the method proposed in this section was implemented on the GPU using nVIDIAs Compute Unified Device Architecture (CUDA). However, detailed explanations are only given for the splitting and, in particular, the merging process as these are parts of the contributions of this thesis. For more detailed insight on how to implement particle-based fluid simulations on GPUs the interested reader is referred to [11, 14].

3.4.1 Preliminaries

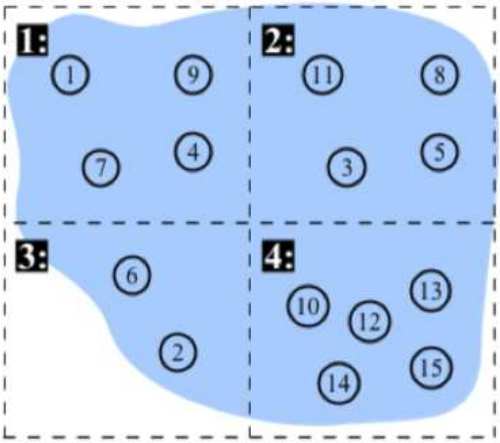
In CUDA tasks are implemented as *kernels*, which are functions of the slightly extended C Programming language *C for CUDA*. Kernels are then executed in parallel by multiple CUDA threads. For each operation a particle is assigned to a thread. Data for particle quanties is stored as linear arrays. This data is referenced by the particle's index. These particle indices are stored in arrays. In the following discussion, the array that stores indices of low-resolution particles referred to as L whereas the array that stores the high-resolution particles is named H.

Particle data is commonly accessed through *global memory*, which resides in the memory of the graphics card (DRAM). Access to global memory is costly. However, it is possible in CUDA to perform coalesced memory access, thus reducing the total number of read and write operations. For this reason, particle data is reordered at the beginning of each time step, so that data of particles that are close to each other in space is adjacent to each other in the linear arrays. Particle indices are then simply stored in ascending order from 0 to $n - 1$, where n is the number of particles in either the low or high resolution domain. Further, to increase the coherence in memory, particle data is stored as a *structure of array*.

For fast neighbor lookup during the SPH computations the approach of Green [11] is adopted. This approach uses a uniform grid to partition the simulation space into smaller grid cells. Each grid cell has a corresponding index according to which particles are sorted. Additional arrays that store references to the particle index array for the first and last particle in a grid cell are then used for determining a particles neighbor. This work follows this approach, however, two uniform grids for the low and high-resolution domain are used.

3.4.2 Splitting

The GPU implementation of splitting in this work is straightforward and essentially follows the procedure described in Section 3.2.2. Each thread checks if its particle can be split. If so, eight high resolution particles are inserted to H and initialized. During initialization these particles are positioned around a cube that is centered around the low-resolution particle. The velocity is simply inherited from the parent and their blending coefficient is set to zero. Lastly, the states of



numParticles	cellOwner	newLowID
4	1	5
4	3	6
2	2	-1
5	10	-1

Figure 6. Detecting particles for merging: The domain is subdivided by a grid. For each grid cell the amount of particles that can be merged is counted. Then one high-resolution particle to be responsible for the cell is chosen and, if possible, the id of the to be inserted low-resolution particle is assigned to the cell.

the high-resolution particles are set `insert` and the state of the low-resolution particle is set to `delete`.

3.4.3 Merging

The implementation proposed in this work uses a uniform grid to separate the simulation domain in order to find high-resolution particles that can be merged (see Figure 6). Examining the simulation domain per grid cell has the advantage that particles within the cell are likely to be arranged properly for merging (see Section 3.2.3). Furthermore, is it easier to search for high-resolution particles in a grid cell as compared to checking the neighbors of each. Since exactly eight particles have to be merged, the spatial increment of the grid Δx is chosen depending on the volume that eight high-resolution particles usually occupy, that is

$$\Delta x \propto (8V_H)^{\frac{1}{3}} \tag{33}$$

where $V_H = m_h/\rho_0$ is the volume a high-resolution particle occupies when it is at rest.

In case eight particles that are eligible to merge are found for a grid cell, one particle (thread) is delegated to add the index of the to be inserted particle to the array that stores all active low-resolution particles L. For this purpose two arrays of the size of the number of grid cells are needed: one to store the number of particles that can be merged per cell and one to save the id of the particle that is delegated to the cell. These particles are referred as to `numParticles` and `cellOwner` in the following discussion (see also Figure 6).

For every grid cell that contains high-resolution particles, the actual merging process is carried out in three steps, which can be briefly described as follows

1. Count all particles that should be merged, and determine the `cellOwner`.
2. If merging is possible, create a new low-resolution particle and determine its index in L.
3. If a new particle is created, initialize its position and velocity by taking the average over all eight high-resolution particles.

The first step is explained in more detail by the following pseudocode [2]

```
1  if merge(i) AND s_i = default then
2      h← computeHash(i);
3      atomicAdd(numParticles[h], 1);
4      atomicMin(cellOwner[h], i);
   endif
```

In the first line the thread checks if the particle in `default` should be split (according to the criteria in Section 3.3). If the particle qualifies for merging, its hash (that is, the index of its grid cell is determined) is computed according to the following equation

$$h = i + i_{max}(j + j_{max}k) \tag{34}$$

where $(i, j, k) = \lfloor \frac{1}{\Delta x}(\mathbf{x}_i - \mathbf{x}_s) \rfloor$ are the grid coordinates of the particles, i_{max}, j_{max} and k_{max} are the number of grid cells in x, y and z direction and $\mathbf{x}_s$ is the origin

[2] This and the following two algorithms are carried for each high-resolution particle i.

of the grid. Finally, the number of mergable particles and the cell owner of the corresponding cell is determined. Atomic operations are used for this in order to avoid race conditions.

In the second step for suitable grid cells, a new low-resolution particle is created and marked for insertion, as follows:

```
1  h← computeHash(i);
2  if numParticles[h] = 8 then
3      if cellOwner[h] = i then
4          newLowID ← atomicAdd(dParticleLowID, 1);
5          lowIDs[h] ← newLowID;
6          createNewParticle(newLowID);
       endif
   endif
7  else
8      lowIDs[h] = -1;
   endif
```

In the second line, each thread checks if exactly eight mergeable particles are contained in the grid cell. If this is the case, a new id for the low- resolution particle is created by adding one to the current number of low- resolution particles (line 3). Subsequently, this id is stored in `lowIDs` for further use. However, if there were not eight mergeable particles, -1 is stored for this cell, indicating that there is no merging carried out for this cell. The function `createNewParticle(newLowID)` sets the position and velocity of the newly inserted particle to zero. Further this particles' state is set to `insert` and its blending coefficient is assigned 0.

The last step initializes the position and velocity of the newly created particle, as follows

```
1  if s_i = default then
2      h ← computeHash(i);
3      newLowID ← lowIDs[h];
4      ownerID ← ownerID[h];
5      if newLowId = -1 then
6          updatePositionAndVelocity(newLowID, i);
7          if OwnerID = i then
8              add(L, newLowID);
           endif
9          s_i ← delete;
       endif
   endif
```

In line 3, for all high-resolution particles the id of the new low-resolution particle they merge into is retrieved. If this id is valid, the merging high-resolution particles add their position and velocity to the new low-resolution particle according to Equations (28, 29) and set themselves to the `delete` state. Finally, the thread with the `ownerID` adds the particle to the list of all active low-resolution particles L.

3.5 Rendering and Surface Extraction

This section briefly discusses the rendering algorithm employed in this thesis and how it is used to extract surface particles.

3.5.1 Screen Space Fluid Rendering

Screen space fluid rendering as proposed by [31, 12] is a popular, efficient and simple to implement method to render particle-based fluids. As outlined in Figure 7, the method takes the particle positions as an input and consists of at least three render passes. In the first render pass the particles are rendered as spheres to create a depth map of the fluid. The depth map is then blurred by an edge preserving filter to smooth the surface, before it is used to reconstruct the surface normals in screen space in a final compositing pass. In the second pass, a thickness map is created to determine the transparency of the fluid. If the fluid is part of

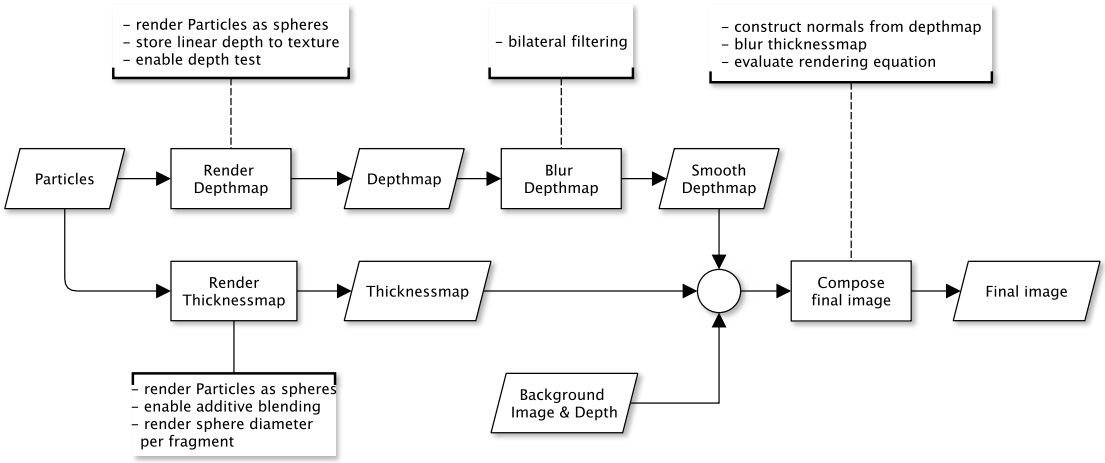


Figure 7. The Screen Space Rendering Process

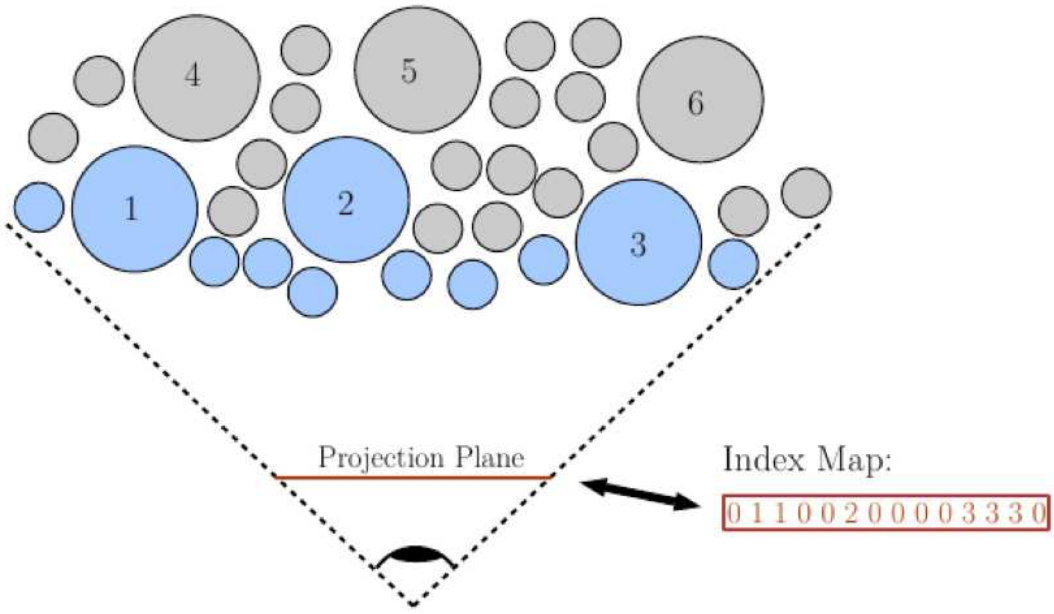


Figure 8. Screen Space Surface Particle Extraction.

a more complex scene, a pass to render the fluids environment has to be carried out as well to create the final image.

3.5.2 Screen Space Surface Extraction

To extract only surface particles that are visible to the camera this work suggests rendering the particle indices to an additional render target, the *index map*, as illustrated in Figure 8. In a following step particles, whose indices are in the index map are flagged as surface particles in parallel using CUDA.

3.6 Summary

This section explained the multi-resolution method proposed in this work. A simple particle blending approach was introduced and numerical instabilities due to this method were addressed by limiting a particles motion according to the CFL condition. Further, resampling criteria for extracting turbulent surface regions were discussed by using the wavelet decomposition of the velocity and screen space rendering. Additionally, a GPU implementation of particle merging was presented.

The method presented in this chapter can be summarized by the following algorithm, where L^n refers to the array of active particles at the current and L^{n+1} refers to the active particle array at the next time-step.

while *animating* **do**
  foreach *particle* $i \in L^n \cup H^n$ **pardo**
    compute density ρ_i *(cf. Eq. 24)*
  end
  foreach *particle* $i \in L^n \cup H^n$ **pardo**
    compute acceleration *(cf. Eq. 24)*
    compute energy *(Eq. 31)*
  end
  foreach *particle* $i \in L^n \cup H^n$ **pardo**
    update position and velocity *(cf. Eq.26)*
    update blending coefficient *(Eq. 23)*
    if $\Lambda_i = 1$ **then**
      $s_i \leftarrow$ `default`;
    endif
    if $\Lambda_i > 0$ **then**
      add($L^{n+1}/H^{n+1}, i$);
    endif
  end
  foreach *particle* $i \in L^n$ **pardo**
    if $s_i =$ ***default*** $\wedge$ ***split(i)*** *(cf. Sec. 3.3)* **then**
      `insert(`H^{n+1}, i`);` *(cf. Sec. 3.4.2)*
      $s_i \leftarrow$ `delete`;
    endif
  end
  foreach *particle* $i \in H^n$ **pardo**
    `merge(i);` *(cf. Sec. 3.4.3)*
  end
endwhile

4. Experimental Results

This section applies the proposed method to a test scenario and compares the computational time to the standard SPH algorithm as described in Section 2.4. The visual result is given in Appendix C.

4.1 Experimental Setup

The scene used in this experiment consists of a block of water that undergoes gravitation on one side of a box shaped canvas (*dambreak*). An additional pillar serves as an obstacle. See Figure 9.

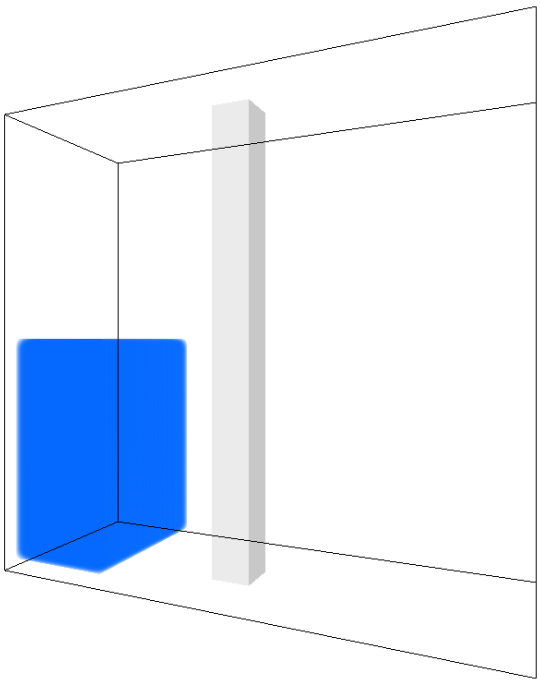


Figure 9. The initial scene

The experiment consists of four runs that simulate the fluid over two seconds of time with different configurations. The visual results of this experiment are given in Appendix C. The simulation time for all configurations is compared to

each other in the following subsection.

The four runs have the following configuration:

Low-resolution simulation 35k particles, splitting and merging disabled

High-resolution simulation 280k particles, splitting and merging disabled

Multi-resolution simulation (no merging) initially 35k particles, splitting enabled, merging disabled

High-resolution simulation (with merging) initially 35k particles, splitting and merging enabled

The specifications of the system this experiment was carried out on are given in Table 1.

CPU	Intel Core i7 3.20 GHz
RAM	16 GB
GPU	NVIDIA GeForce GTX 670
VRAM	2048 MB
OS	Windows 7
API's	CUDA 5.0, OpenGL 3.3 Core Profile (GLSL 3.3)

Table 1. Specifications of the test environment

The choice of physical constants is listed in Table 2.

Constant	**Symbol**	**Value**
Gravitational Acceleration	$\mathbf{g}$	9.81 m/s^2
Rest Density	ρ_0	1000 kg/m^3
Viscosity	μ	5.0 $kg/s \cdot m$
Tait Coefficient	B	30 Pa
Average particle neighbors	$\overline{n}$	35
Blend Increment	$\Delta\Lambda$	0.02
Energy Threshhold	E^θ	$300J$

Table 2. Choice of physical constants.

4.2 Results

Figure 10 shows a comparision on how much simulation time was needed to simulate the fluid over a time span of 2000 ms. The x-axis shows this simulated real time in ms. For all simulations a time step $\Delta t = 1\ ms$ was chosen except for the low-resolution simulation where a time step $\Delta t = 2\ ms$ was chosen. The y-axis denotes the computational time needed for a certain point in real time. As can be seen from the graph the amount of computational time needed for the low-resolution simulation is about nine seconds, whereas the high-resolution simulations needs about 95 seconds. Both multi-resolution simulations take about the same time, 41 seconds.

Figure 11 shows the number of low- and high-resolution particles in the system for every time step of the multi- resolution simulation for the case where merging is disabled. The number of low-resolution particles at the end of the simulation is 28k, whereas the number of high-resolution particles is 65k.

Figure 12 shows the number of low- and high-resolution particles in the system for every time step of the multi-resolution simulation for the case where merging is disabled. The number of low-resolution particles at the end of the simulation is 31k, whereas the number of high-resolution particles is 38.5k. Also, the total number of high-resolution particles never exceeds 40k.

Figures 13 - 16 of Appendix C show the visual results for each run.

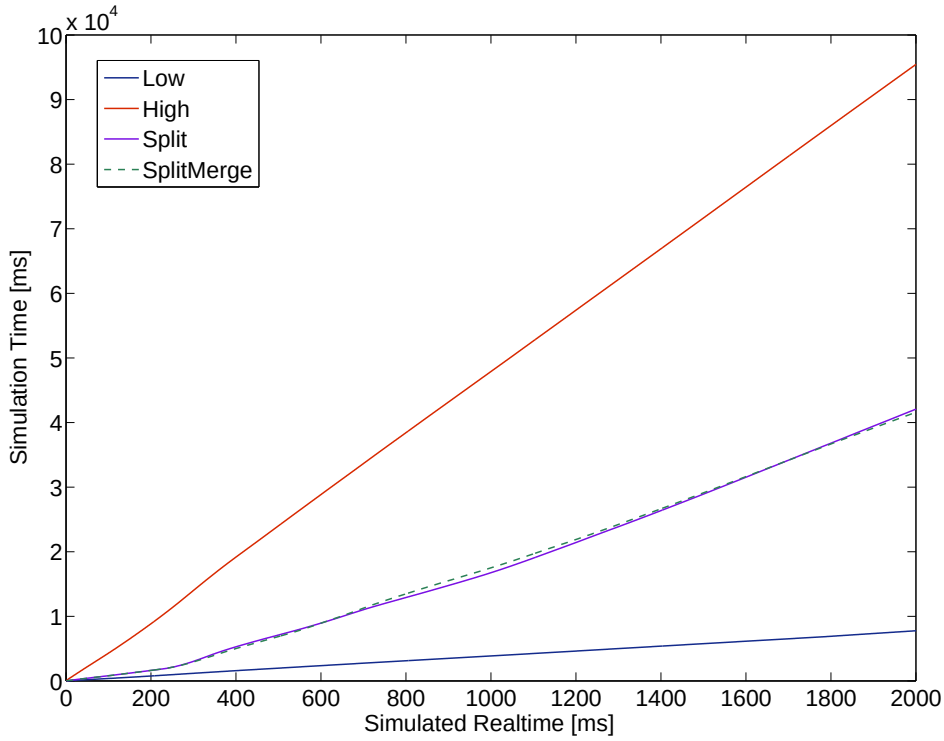


Figure 10. Comparision on how much simulation time is needed to simulate 2000 ms for every run.

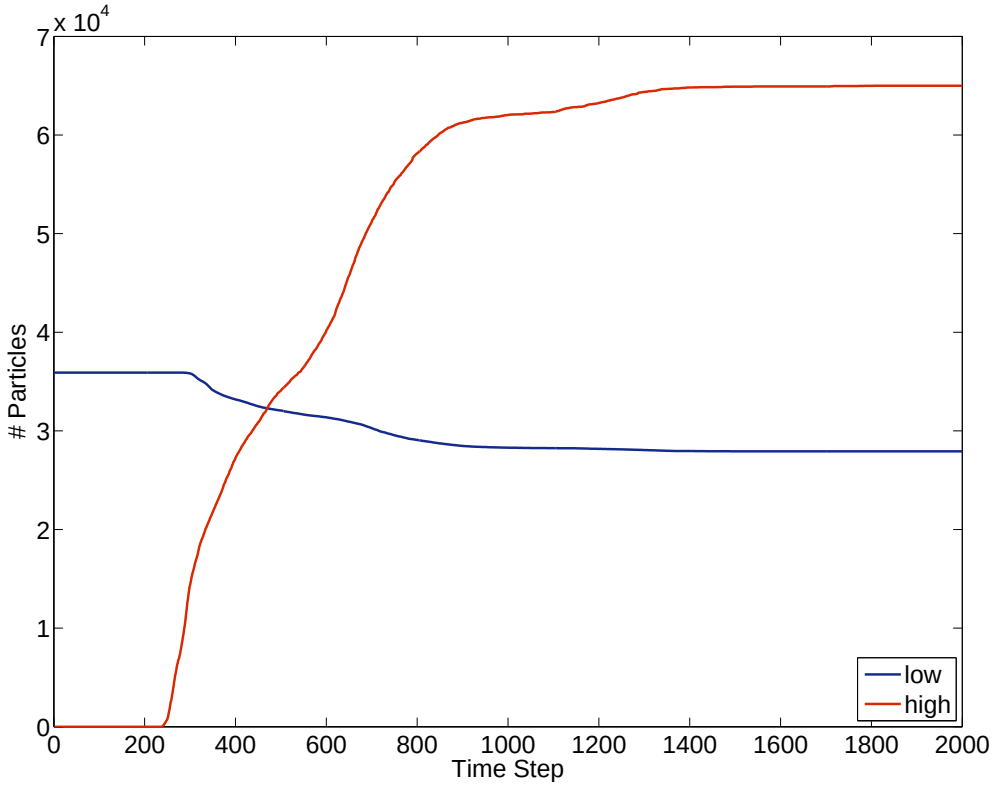


Figure 11. Number of low- and high-resolution particls without merging.

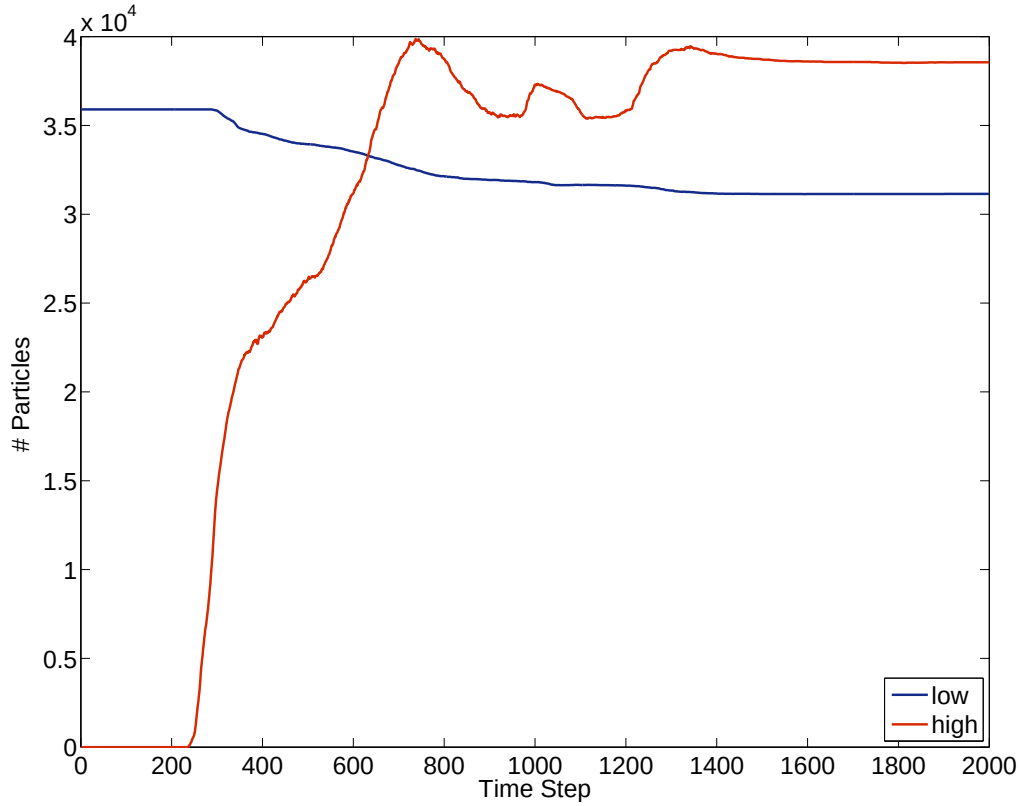


Figure 12. Number of low- and high-resolution particls with merging.

4.3 Summary

In this section the proposed method was tested and compared for a dambreak scenario that had an additional obstacle in form of a pillar. Timings for that experiment have shown that the proposed method is more than two-times faster than the high-resolution method in this scenario. However, no speed-ups could be achieved by merching particles.

5. Conclusion & Future Work

In summary, this work proposed a two-scale particle-based fluid simulation method. The goal was to achieve similar visual qualities of a high resolution simulation by allocating more computional resources to visually important parts of a low-resolution fluid while saving computational time for the rest of the fluid. To achieve this visually important areas were identified as surface areas with high turbulent energy using the wavelet decomposition of the velocities of the fluid. In order to split and merge particles while still preserving large time steps a simple blending-based method that trivially maps to graphics processing units was proposed. The experiment conducted in this work suggested that two times faster simulation times are possible while still preserving similar visual qualities. Based on this results, it can be concluded that multi- scale methods can lead to performance gains, when implemented on GPUs, which can lessen the duration of design cycles as well bring more visual fidelity to real-time applications.

As a major downside the considerable increase of memory due to auxiliary structures should be mentioned. In particular, the use of additional memory for neighbor queries due to the extra resolution is undesirable. This is especially severe, if a high amount of the simulation domain is not needed due to evironmental restrictions. In order to address this problem the usage of more memory efficient data structures such as octrees and their implementation on GPUs as described in [17] can be investigated in future research. Additional memory requirements are due to the auxilliary arrays for the blending coefficients, particle states and merging. Furthermore, the proposed method has been tested for one particular scenario, different experiments with varying scene complexities should be carried out to find out where the proposed ideas are applicable to get performance gains. It would also be interesting to see if the ideas presented here can applied to the recently published and very popular method of Macklin and Müeller [20].

Acknowledgements

First of all, I would like to thank Professor Hirokazu Kato for welcoming me to the Interactive Media Design Lab and giving my the unique and exciting chance to complete my master course in Japan. I am grateful for all the help and advice I received during my stay at his laboratory.

Furthermore, I would like to thank Professor Jun Miyazaki for his help during his lectures and his advice for my research.

I am grateful to Professor Makoto Fujisawa for giving me the opportunity to do my research in a computer graphics related field, for his patience and his valuable comments and advice regarding this research.

I am happy to have met Professor Goshiro Yamamoto and Professor Takafumi Taketomi and would like to thank them for the valuable conversations and always helping me out whenever I had a problem.

During my time at NAIST I was fortunate to meet many great international and Japanese students, I am lucky to have you as my friends.

I would like to thank Eiji, Thanh, Professor Ryuzaburo Sugino and all my other friends at the Anan National College of Technology for their constant help and encouragement.

During my master's course I received the JASSO Honors Scholarschip for Privately Financed International Students, as well as the German Academic Exchange Service's One Year Scholarship for Graduate Students. I would like to thank both organizations for their support.

Finally, I would like to thank my family for their understanding and support, I am greatful to have you.

References

[1] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3), July 2007.

[2] Bart Adams and Martin Wicke. Meshless approximation methods and applications in physics based modeling and animation. In *Eurographics 2009 Tutorials*, pages 213–239, 2009.

[3] Markus Becker and Matthias Teschner. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 209–217, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[4] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3), July 2007.

[5] Robert L. Cook and Tony DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, July 2005.

[6] Mathieu Desbrun and Marie-Paule Cani. Space-Time Adaptive Simulation of Highly Deformable Substances. Rapport de recherche RR-3829, INRIA, 1999.

[7] Makoto Fujisawa, Go Mimura, Toshiyuki Amano, Jun Miyazaki, and Hirokazu Kato. A Fast Simulation Method Using SPH and Wavelet for Sub-Particle-Scale Turbulent Flow. pages 67–72, 2011.

[8] R. A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–398, 1977.

[9] Prashant Goswami and Renato Pajarola. Time adaptive approximate sph. In Jan Bender, Kenny Erleben, and Eric Galin, editors, *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS*, pages 19–28, VRIPHYS 2011, December 2011. Eurographics.

[10] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[11] Simon Green. Cuda particles. *nVidia Whitepaper*, 2(3.2):1, 2008.

[12] Simon Green. Screen space fluid rendering for games. Game Developers Conference, 3 2010.

[13] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed particle hydrodynamics on GPUs. In *Computer Graphics International*, pages 63–70, 2007.

[14] Rama C. Hoetzlein. Fluids v.3: A large-scale, open source fluid simulator. Published online at http://fluids3.com, 2012. Released under Z-lib license.

[15] Mickey Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics. Graduate Project, 2006.

[16] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):50:1–50:6, August 2008.

[17] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Octree textures on the GPU. In Matt Pharr, editor, *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 37, pages 595–613. Addison Wesley, March 2005.

[18] L.B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

[19] M. Mler D. Pomeranerts M. Teschner, B. Heidelberger and M. Gross. Optimized spatial hashing for collision detection of deformable objects. *In proceedings of Vision, Modeling, Visualization*, pages 47–54, 2003.

[20] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4):104:1–104:12, July 2013.

[21] J. J. Monaghan. Simulating free surface flows with sph. *J. Comput. Phys.*, 110(2):399–406, February 1994.

[22] J J Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, 2005.

[23] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[24] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007.

[25] NVIDIA. *NVIDIA CUDA Programming Guide 2.0*. 2008.

[26] Jens Orthmann and Andreas Kolb. Temporal blending for adaptive sph. *Computer Graphics Forum*, 31(8):2436–2449, 2012.

[27] Brandon Pelfrey and Donald House. Adaptive neighbor pairing for smoothed particle hydrodynamics. In *Proceedings of the 6th international conference on Advances in visual computing - Volume Part II*, ISVC'10, pages 192–201, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, April 1983.

[29] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. *ACM Trans. Graph.*, 28(3):40:1–40:6, July 2009.

[30] Barbara Solenthaler and Markus Gross. Two-scale particle simulation. *ACM Trans. Graph.*, 30(4):81:1–81:8, July 2011.

[31] Wladimir J. van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 91–98, New York, NY, USA, 2009. ACM.

[32] Y Zhang, B Solenthaler, and R Pajarola. Adaptive sampling and rendering of fluids on the gpu. In *Proceedings Symposium on Point-Based Graphics*, pages 137–146, August 2008.

Appendix

A. SPH Kernel

The following subsections list the SPH kernels used in this thesis.

A.1 The Poly6 Kernel for the Density:

$$W_{poly6}(\mathbf{x}, h) = C \begin{cases} (h^2 - \|\mathbf{x}\|^2)^3 & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

where C equals to $\frac{4}{\pi h^8}$ in 2D and $\frac{315}{64\pi h^9}$ in 3D

A.2 The Spiky Kernel for the Pressure Term:

$$W_{spiky}(\mathbf{x}, h) = C \begin{cases} (h - \|\mathbf{x}\|)^3 & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

$$\nabla W_{spiky}(\mathbf{x}, h) = -3C \frac{\mathbf{x}}{\|\mathbf{x}\|} \begin{cases} (h - \|\mathbf{x}\|)^2 & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

where C equals to $\frac{10}{\pi h^5}$ in 2D and $\frac{15}{\pi h^6}$ in 3D

A.3 The Viscosity Kernel for the Viscosity Term:

in 3D

$$W_{viscosity}(\mathbf{x}, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{\|\mathbf{x}\|^3}{2h^3} + \frac{\|\mathbf{x}\|^2}{h^2} + \frac{h}{2\|\mathbf{x}\|} - 1 & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta W_{viscosity}(\mathbf{x}, h) = \frac{45}{\pi h^6} \begin{cases} h - \|\mathbf{x}\| & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

in 2D

$$W_{viscosity}(\mathbf{x}, h) = \frac{40}{\pi h^2} \begin{cases} -\frac{\|\mathbf{x}\|^3}{9h^3} + \frac{\|\mathbf{x}\|^2}{4h^2} + \frac{1}{6}\log\frac{\|\mathbf{x}\|}{h} - \frac{5}{36} & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta W_{viscosity}(\mathbf{x}, h) = \frac{40}{\pi h^5} \begin{cases} h - \|\mathbf{x}\| & \|\mathbf{x}\| < h \\ 0 & \text{otherwise} \end{cases}$$

B. Estimation of Additional Simulations Parameters

The following subsections describe how the mass and effective radius for all particles were computed.

B.1 Particle Mass

The mass for all particles can be computed as follows

$$m = \frac{\rho_0 V}{n}, \tag{35}$$

where ρ_0 is the rest density of the fluid, V is the (initial) volume of the fluid and n is the number of particles.

B.2 Effective Radius

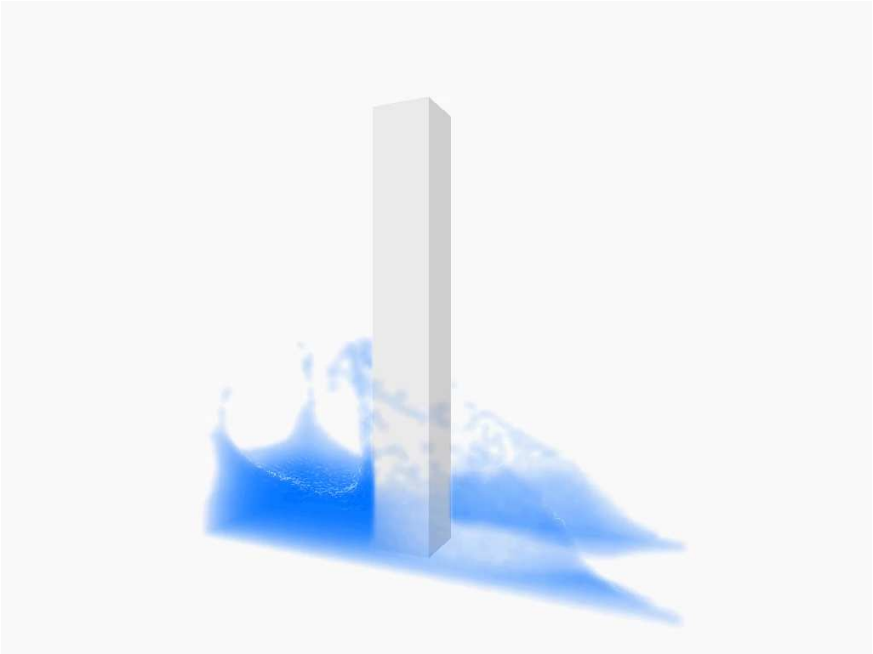
Assuming an average of $\overline{n}$ particles should lay within an effective radius h of a particle occupying approximately a volume of $\frac{4}{3}\pi h^3 \approx \frac{\overline{n}}{n}$, the effective radius for all particles can be computed as follows

$$h = \left(\frac{3\overline{n}V}{4\pi n}\right)^{\frac{1}{3}}, \tag{36}$$

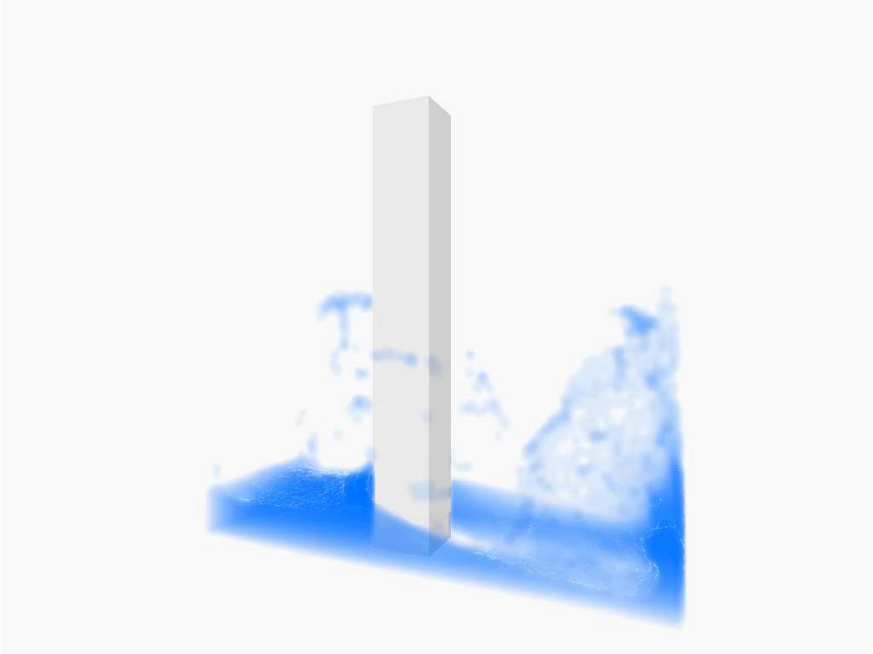
where V is the (initial) volume of the fluid and n is the number of particles.

C. Visual Results

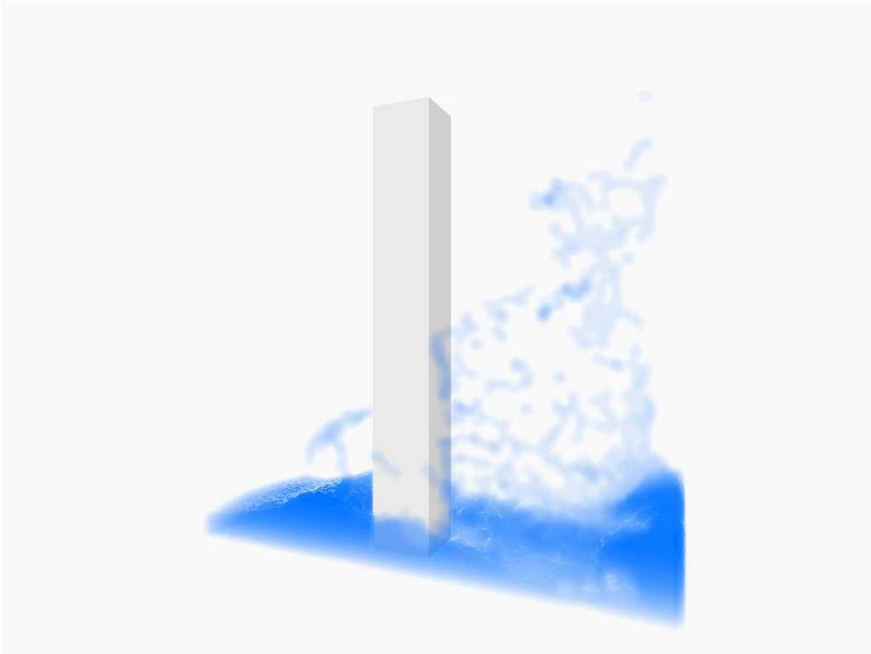
The following figures show the visual results of the experiment described in section 4 for each simulation for time steps 500, 750, 1000, 1025.



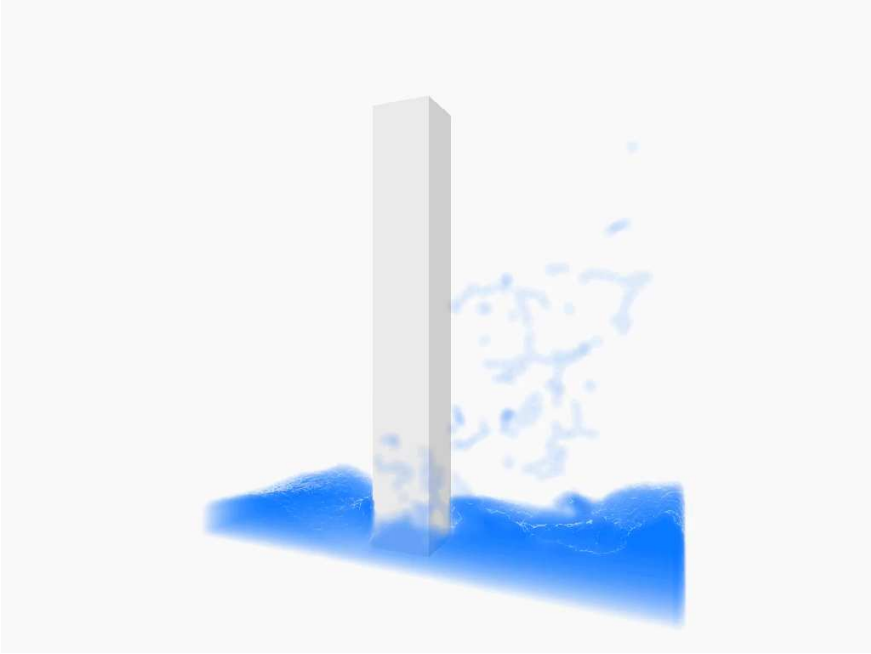
step 500



step 750

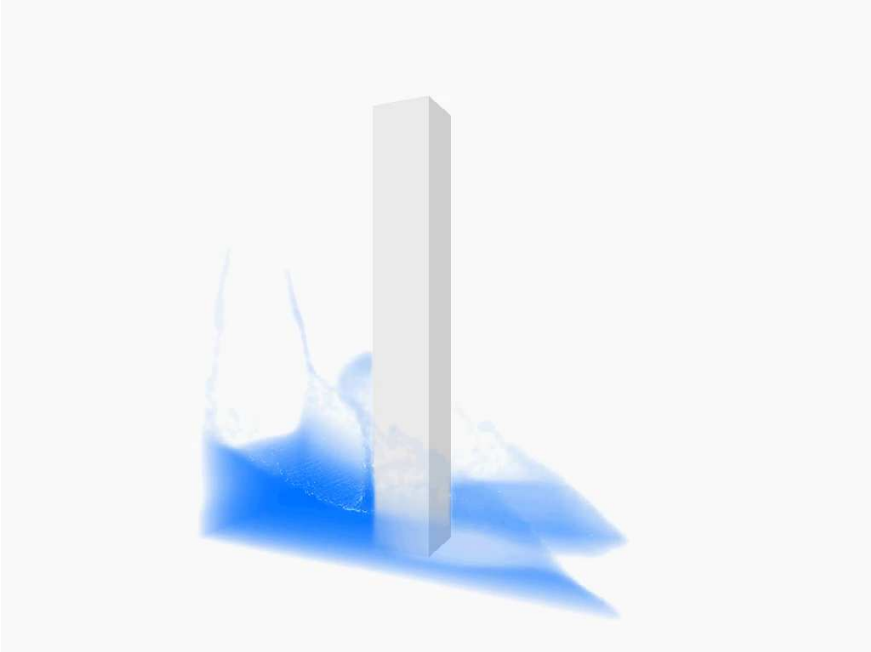


step 1000



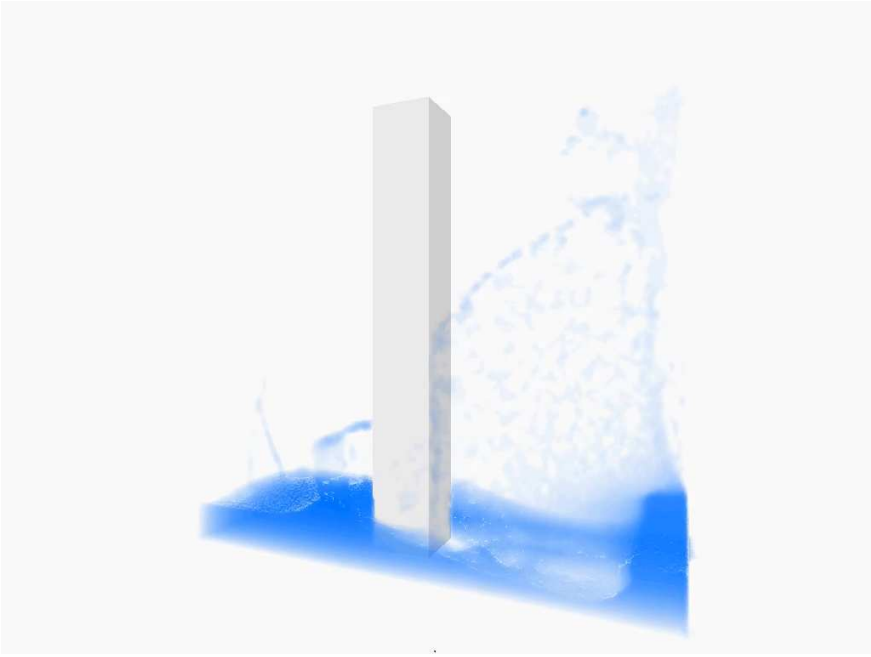
step 1250

Figure 13. Low-resolution simulation results

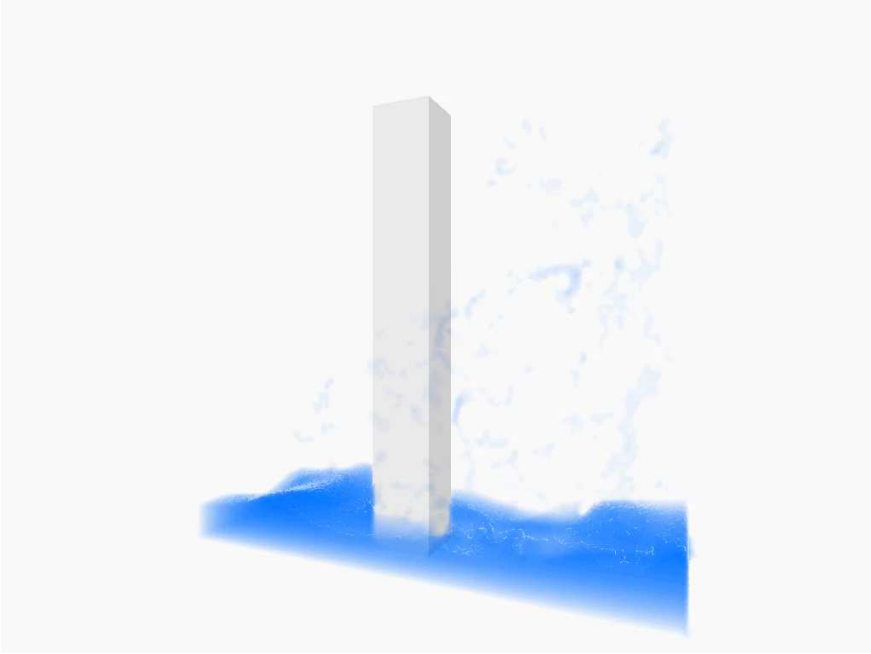


step 500

step 750

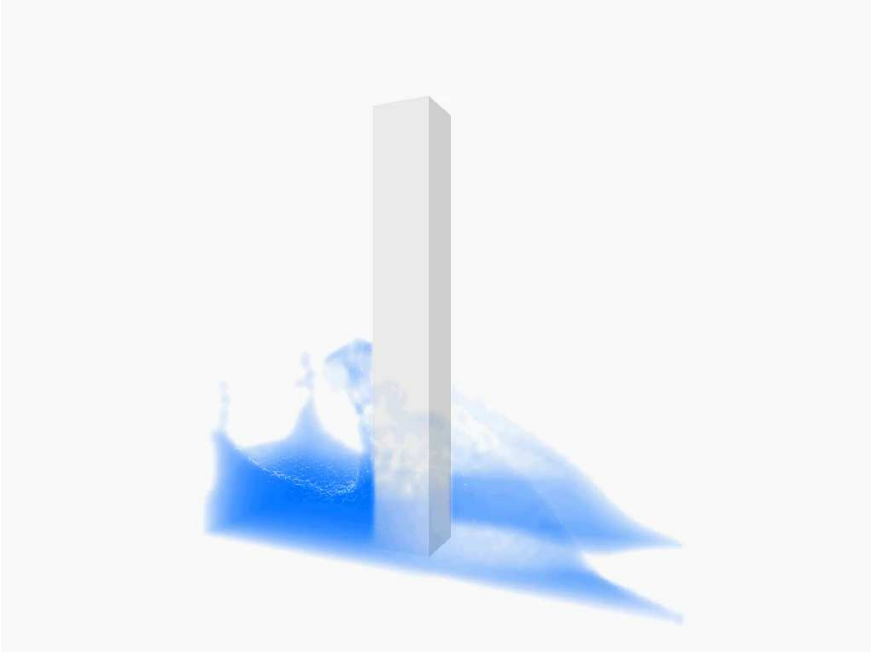


step 1000

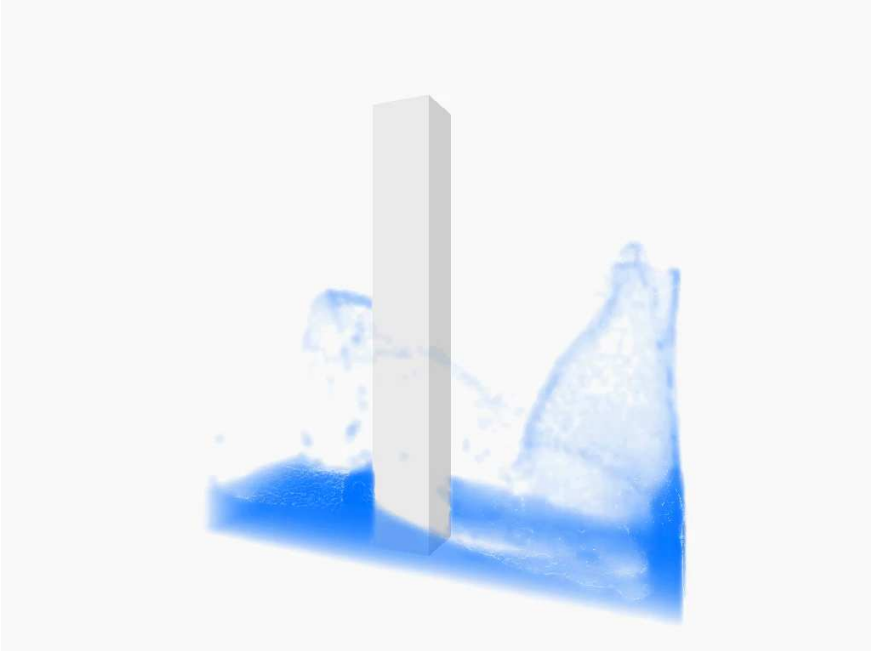


step 1250

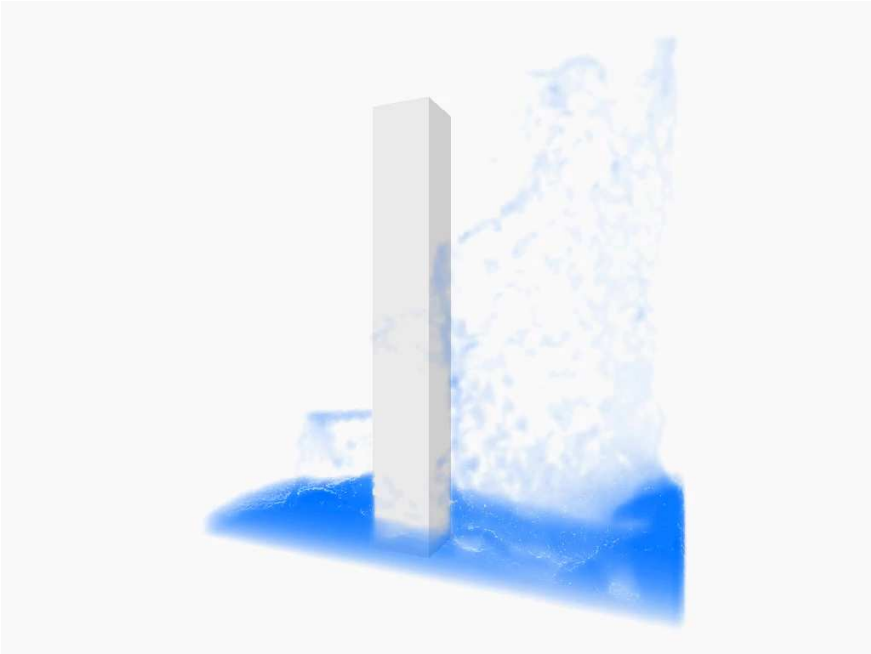
Figure 14. High-resolution simulation results



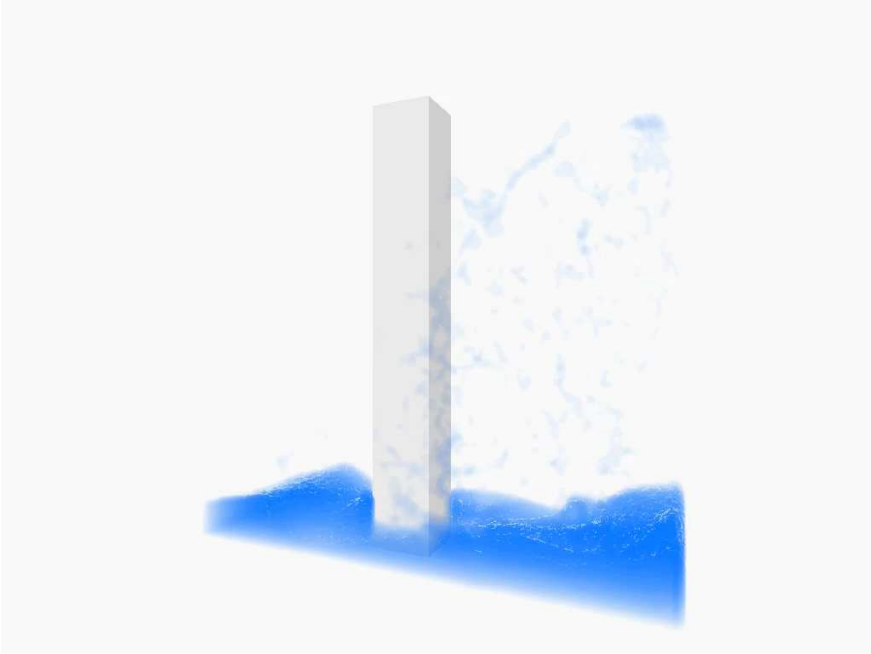
step 500



step 750

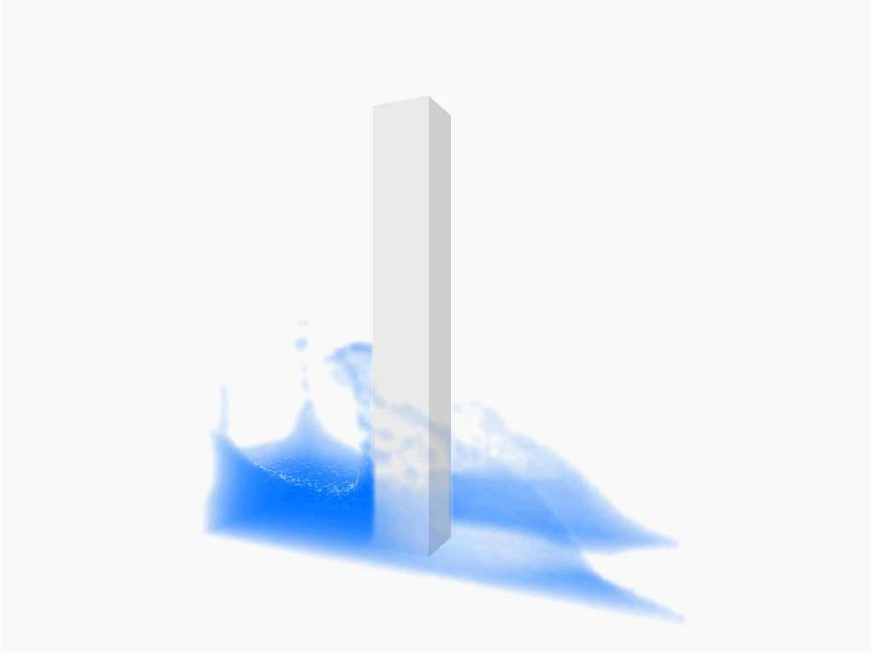


step 1000

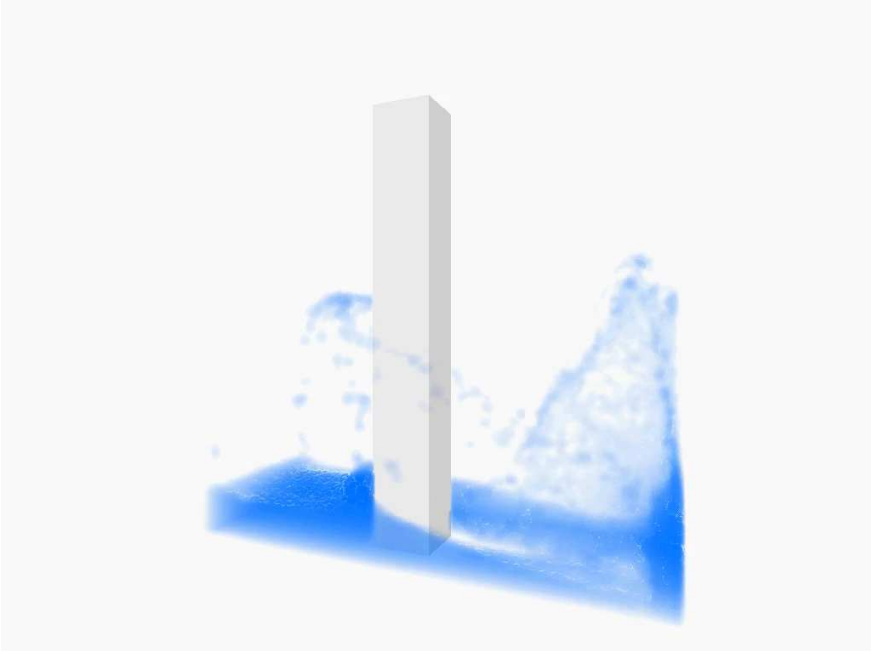


step 1250

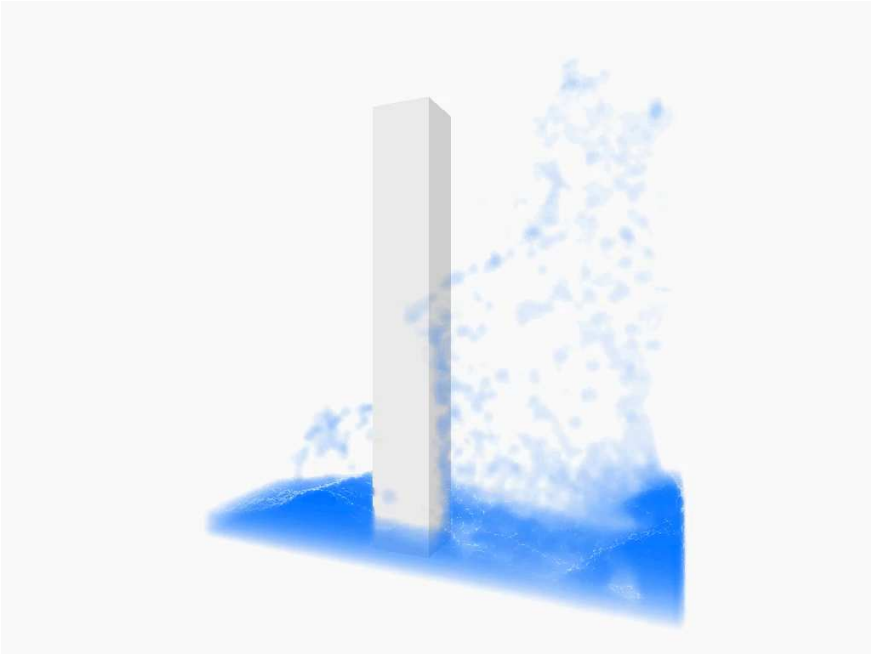
Figure 15. Multi-resolution simulation results (no merging)



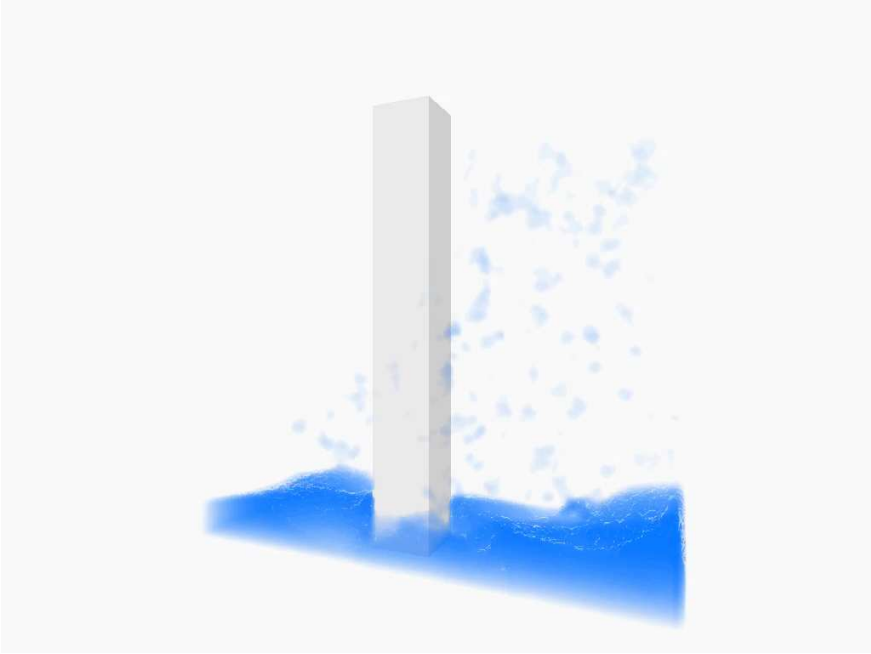
step 500



step 750



step 1000



step 1250

Figure 16. Multi-resolution simulation results (with merging)