Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics

Micky Kelager
micky@kelager.dk

January 9, 2006

DIKU
Department of Computer Science • University of Copenhagen
Universitetsparken 1 • DK-2100 Copenhagen • Denmark

Abstract

Computational fluid dynamics is a hot topic in Computer Graphics. The capability to reproduce realistic fluids numerically has gained an increased interest the last decade. Grid-based methods have been favored the most to solve the mathematical equations for fluid flow, but often they lack the ability to create interactive fluid simulations together with detailed fluid surfaces. Interactive fluid dynamics is of essential interest in real-time applications, such as computer games or virtual surgery simulators. Using the smoothed particle hydrodynamics (SPH) method, we have implemented a stable particle-based approach to solve the motion of interactive fluids. With focus on the simulation part we provide a thorough insight of the mathematical theory of particle-based fluids. Detailed real-time results are obtained for small-scale fluid simulations of water and viscous mucus, but the interactivity is sacrificed when near-incompressibility is enforced.

Contents

1 Introduction 1
1.1 Software Solutions 1
1.2 Goals 3
1.3 Overview 3
2 Classical Fluid Dynamics 4
2.1 The Navier-Stokes Equations 4
2.2 Eulerian Fluids 6
2.3 Summery 7
3 Smoothed Particle Hydrodynamics 8
3.1 Definitions 8
3.2 Smoothing Kernels 11
3.3 Summery 13
4 Lagrangian Fluid Dynamics 14
4.1 Mass-Density 16
4.2 Internal Forces 17
4.2.1 Pressure 17
4.2.2 Viscosity 20
4.3 External Forces 23
4.3.1 Gravity 23
4.3.2 Buoyancy 23
4.3.3 Surface Tension 24
4.3.4 User Interaction 26
4.4 Collision Handling 26
4.4.1 Collision Detection 27
4.4.2 Implicit Primitives 28
4.4.3 Collision Response 32
4.4.4 Discussion 34
4.5 Numerical Time Integration 36
4.5.1 The Implicit Euler Scheme 36
4.5.2 The Verlet Scheme 36
4.5.3 The Leap-Frog Scheme 37
4.5.4 Discussion 38
4.6 Summery 38
5 Implementation 40
5.1 Fast Nearest Neighbor Search 41
5.1.1 Spatial Hashing 41

5.1.2 Spatial Particle Queries ... 42
5.2 Incompressibility ... 43
5.2.1 Discussion ... 44
5.3 Physical Parameters ... 45
5.3.1 Fluid Volume and Particle Mass ... 45
5.3.2 Smoothing Kernel Support Radius ... 45
5.3.3 Time Integrator and Time Step ... 47
5.3.4 Gas Stiffness and Rest Density ... 47
5.3.5 Viscosity Coefficient ... 48
5.3.6 Surface Tension and Threshold ... 49
5.4 Fluid Materials ... 50
5.4.1 Water ... 50
5.4.2 Mucus ... 51
5.4.3 Steam ... 51
5.5 Rendering ... 52
5.6 The Lagrangian Fluid Method ... 53
5.6.1 Initialize SPH System ... 53
5.6.2 Compute Density and Pressure ... 54
5.6.3 Compute Internal Forces ... 54
5.6.4 Compute External Forces ... 54
5.6.5 Time Integration and Collision Handling ... 55
5.6.6 Render Particles ... 55
5.7 Discussion ... 55
5.8 Summery ... 57
6 Results ... 59
6.1 Fluid Properties ... 59
6.2 Fluid Flows ... 63
6.3 Sanity and Stability ... 68
6.4 Performance Tests ... 71
6.5 Issues and Challenges ... 73
7 Future Work ... 77
7.1 GPU Utilization ... 77
7.2 Fluid Visualization ... 77
7.3 Incompressible Lagrangian Fluids ... 78
7.4 Advanced Fluid Interactions ... 78
8 Conclusion ... 79
8.1 Contributions ... 80
References ... 81

Preface

This is a graduate project in computer science at DIKU, the Department of Computer Science, University of Copenhagen. The project caters for people interested in a detailed particle-based description of a numerical solution to interactive fluid simulations. This report is written by Micky Kelager, DIKU. Supervisor is assistant professor Kenny Erleben, DIKU.

Contributions

Based on the previous work done in the field of particle-based fluid simulations, this project contributes a thorough insight of the mathematical theory of particle-based fluid motion, physically correct fluid parameters, stable collision handling between fluid particles and implicit primitives, visual analysis of fluid flows using SPH in Computer Graphics, and a complete open-source implementation of particle-based fluids.

Prerequisites

The reader is assumed to have passed the undergraduate studies in computer science or equivalent, and to possess knowledge in linear algebra and calculus corresponding to the undergraduate courses in mathematics. A basic knowledge in classical mechanics is also assumed, i.e. the physical laws of motion, and an interest in physics-based animation will do no harm.

Acknowledgements

Personally, I wish to thank my girlfriend for accepting and supporting the endless hours I have spent in front of my computer completing this project. Thanks to Theo Engell-Nielsen for reviewing this work and for providing useful comments and suggestions. Also thanks to Henrik Dohlmann for following up on the standard appliances of the implementation in OpenTissue. Finally, huge thanks to my supervisor, Kenny Erleben, for all the hours spent helping me complete the project and for providing graceful guidelines, theoretically and in practice. Always available at days, nights, weekends, and holidays, your guidance has been most appreciated.

Micky Kelager, DIKU.
January 9, 2006.

1 Introduction

Today computational fluid dynamics (CFD) is a hot topic in Computer Graphics. Researchers concentrate on developing new and better methods to simulate and visualize fluids. Common for them all is the mathematical equations that describe the motion of fluids. A fluid is commonly said to be one of the hardest phenomena to simulate realistically, and even harder is the ability to simulate detailed fluids interactively. Offline simulations are the most common, where a vast majority of either particles, grid cells, or a hybrid of grid and particles are being used throughout the simulation. High resolution images can be produced when the fluid is visualized using rendering techniques such as ray tracing.

Grid-based fluid implementations have been favored the most in Computer Graphics for the last decade. When it comes to real-time detailed fluid simulations in 3D a grid-based solution is not optimal. Recently, particles have been introduced into CFD in Computer Graphics. It is out understanding that particles have proven to be a good choice for detailed small-scale fluid simulation that allow for interactive rates. Interactivity is required in areas of computer games, real-time surgery simulators, and as a draft version of an animation that can give the animator a good understanding of the final result. We are interested in a fast and stable particle-based method for fluid simulations to be integrated into OpenTissue, which is an academic open-source project for physics-based animation and surgery simulation.

Animation of fluids consists of two equally interesting parts; simulation and visualization. We believe that much work on fluid visualization has already been achieved in Computer Graphics, and for that reason this report will focus on the simulation part. That is, the mathematics and dynamics required to describe the complex motion of fluids.

1.1 Software Solutions

In this section we will briefly describe two software solutions that provide particle-based fluids. Both solutions are commercial products that aim at the professional end user.

RealFlow^3 by Next Limit Technologies [30] is a complete standalone fluid application that supports different types of fluids, e.g. gasses and liquids. RealFlow^3 calculates the particle motions to simulate fluid flows that are designed by an end user. Interactions with the fluid must be performed offline, i.e. from a key-frame animation. The pre-computed particle motions, together with other particle attributes, can be imported into various 3D visualization applications for final rendering of the fluid surface. Figure 1.1 depicts a screenshot from RealFlow^3 performing a computation of a simple water simulation of approximately $3,000$ particles. The particles are emitted from a circular source, and rendered as dots with no volume. The computation time increased quickly for each frame, and the

first 150 frames took 86 seconds to compute in average, i.e. a rate of 1.74 frames per second. RealFlow³ works offline even for small-scale fluid simulations.

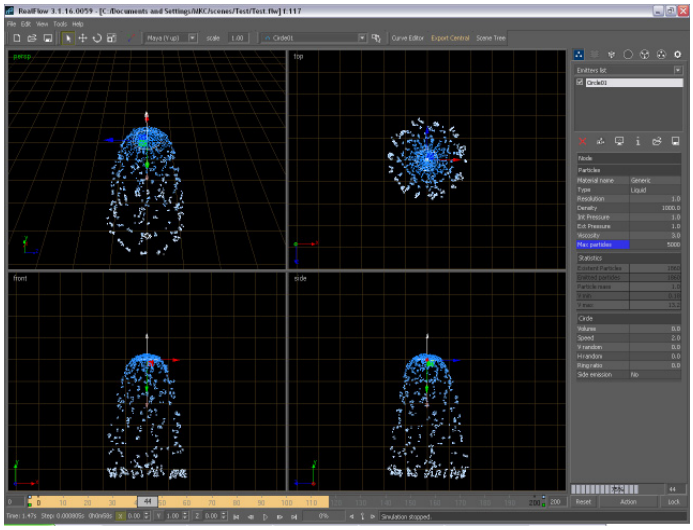


Figure 1.1 Screenshot from RealFlow³ by from Next Limit Technologies.

PhysX by AGEIA [29] provides interactive fluid simulations in real time. However, additional hardware support must be present if real-time rendering of the free fluid surface is required. AGEIA is the manufacturer of the new physics processing unit (PPU) that aims at easing the CPU by taking over the computational work from the demanding physics calculations. A surface extraction algorithm is implemented on the PPU that provides real-time triangulation of isosurfaces, e.g. the free surfaces of liquid fluids. The new PPU boards are only available on the PC, as it is assumed that next generation consoles, e.g. Xbox360 and Sony Playstation 3, have enough processor power to perform the same operations without support of additional hardware. Figure 1.2 depicts a screenshot from a PPU supported fluid simulation.

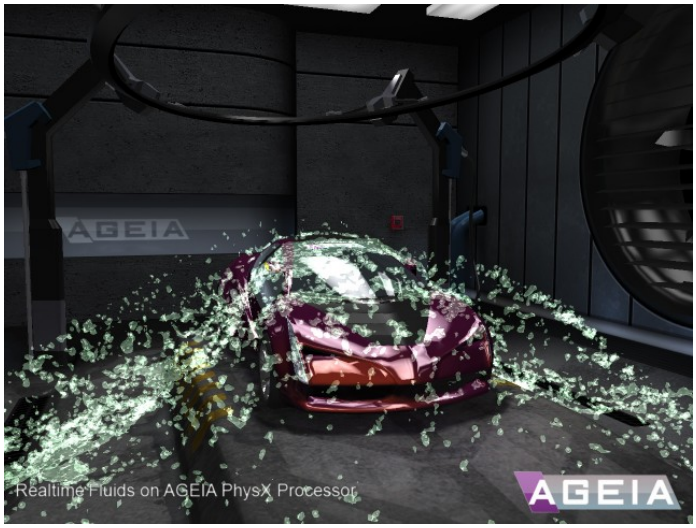


Figure 1.2 Screenshot from PhysX by AGEIA for a real-time fluid animation.

Recently AGEIA has released a PhysX SDK, formally known as NovodeX, which supports fluids in hardware using the PPU, but also in software. The SDK does not reveal any details on the implementation of the fluid solver. At this stage we have not been able to implement any interesting fluid effects using the SDK, but we will assume that the software can create convincing fluids, as one of the architects behind the implementation is Dr. Matthias Müller, author of various papers on particle-based fluid simulations. As PhysX is a middleware physics engine that targets computer games, it has been designed with performance in mind. It is not uncommon for commercial physics engines to hack the physical properties to obtain stability very quickly.

1.2 Goals

The primary goal of this work is to develop a particle-based method that is suitable for simulating fluids at interactive rates. Another goal for this work is to use correct physical quantities and parameters for the simulations. Physically correct parameters are important for potential users and programmers of our fluid solver, as they can rely on reality rather than on fiction.

To achieve the goals satisfactory it is important to understand the theories of fluid dynamics. Readers of this report should be able to learn about our particle-based approach of fluid simulation, and to comply with this reason we will describe the details of the mathematical and physical toolboxes that render interactive fluids possible.

1.3 Overview

In the next chapter we will briefly sum up the previous work done on grid-based fluids in Computer Graphics, which also includes the main theorem of fluid motion. In Chapter 3 we introduce the reader to smoothed particle hydrodynamics, a mathematical toolbox that makes Lagrangian fluids possible for our purpose. Chapter 4 describes the dynamics of a particle-based fluid simulation in full, and implementation details along with physical secrets are reviled in Chapter 5. The results from our work are presented in Chapter 6. The topics presented in this work serve as a basis on particle-based fluids, and Chapter 7 highlights some of the interesting improvements and extensions that we believe follow naturally. Finally, we conclude our work In Chapter 8.

2 Classical Fluid Dynamics

In Computer Graphics the motion of fluids is an important issue when simulating everyday phenomena, e.g. rain, mud, pouring water, cigarette smoke, steam, wet foam, ocean waves, etc. In this chapter we will briefly introduce the reader to the famous Navier-Stokes equations of fluid flow, which serve as a basis for this work. To implement the Navier-Stokes equations for fluid motion, two fundamental approaches exist; a grid-based method, also called the Eulerian method, and a particle-based method, known as the Lagrangian method. The Euler grid-based method is by far employed the most in Computer Graphics, and the Navier-Stokes equations have been extended to simulate many interesting effects, including surface tension for animation of water and milk [20], viscoelasticity and elastoplasticity for animation of mucus, pudding, and clay [10], and thermal buoyancy for animation of hot, turbulent gas [9]. The majority of the Eulerian implementations and their extensions work offline, i.e. they do not contribute to an interactive solution.

If the surface of a liquid alone is the interesting part, the most commonly interactive methods used today, e.g. in computer games, are sinusoidal and gravity waves that can be implemented as a vertex shader on a graphics processing unit (GPU) very efficiently. A hybrid between gravity waves and the Navier-Stokes equations is the shallow water equations [18], which can simulate the interactions between dynamic waves and the boundaries at interactive rates.

2.1 The Navier-Stokes Equations

The basic physical quantities of an isothermal, viscous fluid are velocity $\mathbf{u}$, mass-density ρ, and pressure p. The quantities are considered as continuous fields in the fluid. The classical formulation of the motion for incompressible fluid flow over time t is governed by the Navier-Stokes equations [7],

$$\rho\Big(\frac{\partial}{\partial t} + \mathbf{u}\cdot\nabla\Big)\mathbf{u} = -\nabla p + \mu\nabla\cdot(\nabla\mathbf{u}) + \mathbf{f}\,, \tag{2.1}$$

$$\nabla\cdot\mathbf{u} = 0\,, \tag{2.2}$$

where μ is the viscosity of the fluid, and $\mathbf{f}$ is the sum of external force-densities acting on the fluid, e.g. gravity.

The Navier-Stokes formulation of fluid motion is based on a grid structure, see Section 2.2 for further details on grids. This ultimately means that field quantities not only depend on time, but also on grid positions $\mathbf{r}(t)$, which also depend on time. In three-dimensional space the position vector is given by

$$\mathbf{r}(t) = \big(x(t), y(t), z(t)\big)^T . \tag{2.3}$$

Equation (2.1) describes the conservation of momentum, and basically it is Newton's 2nd law for a fluid. The right hand side represents the total force-densities, while the left hand side is the product of mass-densities and acceleration-densities. For a fluid the acceleration-density is the full time derivative of the velocity field. Using the chain rule on the velocity field yields,

$$\begin{aligned}
\tfrac{d}{dt}\mathbf{u}(t,\mathbf{r}(t)) &= \tfrac{\partial \mathbf{u}}{\partial t} + \tfrac{\partial \mathbf{u}}{\partial x}\tfrac{\partial x}{\partial t} + \tfrac{\partial \mathbf{u}}{\partial y}\tfrac{\partial y}{\partial t} + \tfrac{\partial \mathbf{u}}{\partial z}\tfrac{\partial z}{\partial t} \\
&= \tfrac{\partial \mathbf{u}}{\partial t} + \left[\tfrac{\partial \mathbf{u}}{\partial x}.\tfrac{\partial \mathbf{u}}{\partial y}.\tfrac{\partial \mathbf{u}}{\partial z}\right]^T \cdot \left[\tfrac{\partial x}{\partial t}, \tfrac{\partial y}{\partial t}, \tfrac{\partial z}{\partial t}\right]^T \\
&= \tfrac{\partial \mathbf{u}}{\partial t} + \left(\mathbf{u} \cdot \left[\tfrac{\partial}{\partial x}.\tfrac{\partial}{\partial y}.\tfrac{\partial}{\partial z}\right]^T\right)\mathbf{u} \\
&= \Big(\tfrac{\partial}{\partial t} + \mathbf{u}\cdot\nabla\Big)\mathbf{u}.
\end{aligned} \tag{2.4}$$

Generically, the rate of change of a continuous, dimensionless field, $\psi(t,\mathbf{r}(t))$, in the fluid that follows the fluid over time, is called the substantial derivative of ψ, and is defined as

$$\frac{d\psi}{dt} = \frac{\partial \psi}{\partial t} + \mathbf{u}\cdot\nabla\psi , \tag{2.5}$$

where $\mathbf{u}\cdot\nabla\psi$ is the advection term, and (2.4) justifies how this term arrives, mathematically. The substantial derivative is the full time derivative of the field ψ in the Eulerian view.

Equation (2.2) describes the conservation of mass for an incompressible fluid, and was originally formulated in Euler's Equations [7]. In general, the mass conservation equation for a compressible fluid is also known as the continuity equation. It is the substantial derivative of the density field, which yields

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0 . \tag{2.6}$$

When the mass-density is constant throughout the fluid, it implies that $\frac{\partial \rho}{\partial t} = 0$ and $\rho\nabla\cdot\mathbf{u} = \nabla\cdot\mathbf{u}$. The fluid is then called incompressible, and we end up with the divergence free velocity field as in (2.2).

2.2 Eulerian Fluids

The Navier-Stokes equations (2.1) and (2.2) formulate the motion of an Eulerian fluid. The fluid is though of as being composed of fluid cells, aligned in a regular grid, each of which contains a number of fluid molecules, or particles. Figure 2.1 depicts a basic layout of a grid-based fluid, which has been reduced to two-dimensions for reasons of clarity. Due to the coarse resolution of the 6×6 sized grid, the whole fluid surface is contained in the same row, which is a common problem for details and visualization.

The Navier-Stokes equations defy a full analytic solution, but can be solved numerically using several steps for each component of the equations. One of the most important steps is the Helmholtz-Hodge Decomposition, which is a mathematical technique that allows the pressure gradient $(-\nabla p)$ and (2.2) to be combined into a single equation. The different steps are solved using explicit, implicit, and semi-implicit integrators. The grid provides a solution to estimate derivatives using a finite difference method (FDM). For theoretical and implementation details on Eulerian fluid see [33, 11, 7].

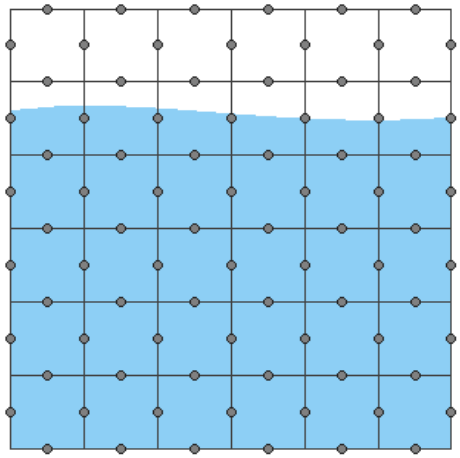


Figure 2.1 Euler grid-based fluid structure in 2D. The discrete velocity field is represented at the dots.

Although the Eulerian method provides a better description of some of the fluid properties, e.g. the mass-density and pressure field, a major disadvantage by the method is the grid itself. The fluid is constrained to stay with the grid. It simply cannot exist outside the grid domain, and thus makes it a difficult problem to let the fluid flow naturally, e.g. when a fluid container fractures. Adaptive grid structures have been used to overcome the fixed grid for different purposes. In [20] an adaptive octree method is introduced to simulate high detailed water and smoke. Another issue is regarding the grid size and memory consumption. In 3D a coarse grid uses an enormous amount of memory to simulate a high resolution fluid. Newer sparse level set data structures can overcome the memory problem of coarse grids. Also a dynamic expansion of the grid domain is no longer a problem, i.e. the grid can grow endlessly. In [13] the run-length encoded (RLE) sparse level set is

presented which is highly scalable. In [26] the dynamic tubular grid (DT-Grid) allows for a very low memory footprint when representing high resolution level sets. In [14] the hierarchical run-length encoded (H-RLE) level set data structure is introduced that combine the best features from the DT-Grid and the RLE sparse level set. The Hamilton-Jacobi partial differential equations (PDEs), can generally be solved using level set methods [27]. The Navier-Stokes equations can be adopted into Hamilton-Jacobi PDEs, thus the new sparse level set structures can be used to simulate fluids with high details and manageable memory consumptions.

Even though special data structures can be used to increase the details and lower the memory consumptions for Eulerian grid-based fluid simulations, grid-based methods still cannot produce interactive results for detailed fluids. In [8] the simulation time for viscous mud is 3 minutes per frame on a grid of $150\times200\times150$ cells, and the computation time is about 4-5 minutes per frame for detailed water and milk with an effective grid resolution of $512\times512\times512$ [20].

2.3 Summery

In this chapter the following important topics are covered/achieved:

- The Navier-Stokes equations for an incompressible, isothermal fluid is based on a grid formulation, which yields
$$\rho\Big(\frac{\partial}{\partial t}+\mathbf{u}\cdot\nabla\Big)\mathbf{u}=-\nabla p+\mu\nabla\cdot(\nabla\mathbf{u})+\mathbf{f}\,,$$
$$\nabla\cdot\mathbf{u}=0\,.$$
- The field quantities for an Eulerian fluid depends on time, t, and grid position, $\mathbf{r}(t)$, e.g. the pressure field, $p(t,\mathbf{r}(t))$.
- The frame computation time for a detailed grid-based fluid simulation in 3D is several minutes, thus the Eulerian fluid method in general does not support an interactive solution.

3 Smoothed Particle Hydrodynamics

The smoothed particle hydrodynamics (SPH) formulation originates from the field of computational astrophysics and is designed for compressible flow problems [21]. The method has been widely used to simulate astrophysical phenomena, where complex problems could be expressed and understood more intuitively. SPH is an interpolation method to approximate values and derivatives of continuous field quantities by using discrete sample points. The sample points are identified as smoothed particles that carry concrete entities, e.g. mass, position, velocity, etc., but particles can also carry estimated physical field quantities dependent of the problem, e.g. mass-density, temperature, pressure, etc. The SPH quantities are macroscopic and obtained as weighted averages from the adjacent particles.

Compared to other well-known methods for numerical approximation of derivatives, e.g. the finite difference method, which requires the particles to be aligned on a regular grid, SPH can approximate the derivatives of continuous fields using analytical differentiation on particles located completely arbitrary. Each particle is thought of as occupying a fraction of the problem space, and to get more accurate weighted quantity averages the sample particles must be dense.

The SPH method was adapted into the Computer Graphics community in [32], where it was used to solve diffusion equations numerically. It has since been extended for various problems, including highly deformable bodies [5], lava flows [34], and computational fluid dynamics [23].

3.1 Definitions

SPH is basically an interpolation method. The interpolation is based on the theory of integral interpolants using kernels that approximate a delta function. The integral interpolant of any quantity function, $A(\mathbf{r})$, is defined over all the space, Ω, by

$$A_I(\mathbf{r}) = \int_\Omega A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)d\mathbf{r}', \tag{3.1}$$

where $\mathbf{r}$ is any point in Ω, and W is a smoothing kernel with h as the width. The width, or core radius, is a scaling factor that controls the smoothness or roughness of the kernel. Section 3.2 concerns smoothing kernels.

The numerical equivalent to (3.1) is obtained by approximating the integral interpolant by a summation interpolant,

$$A_S(\mathbf{r}) = \sum_j A_j V_j W(\mathbf{r}-\mathbf{r}_j,h), \tag{3.2}$$

where j is iterated over all particles, V_j is the volume attributed implicitly to particle j, $\mathbf{r}_j$ the position, and A_j is the value of any quantity A at $\mathbf{r}_j$. Compared to (3.1) the volume for each iterated particle is added to (3.2), which can be justified when A_j is assumed sufficient constant on V_j. The following relation between volume, mass, and mass-density applies

$$V = \frac{m}{\rho}, \tag{3.3}$$

where m is the mass and ρ the mass-density. Substituting the volume for particle j in (3.2) with (3.3) yields the basis formulation of the SPH method, which can be used to approximate any continuous quantity field, and be evaluated everywhere in the underlying space,

$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W\left(\mathbf{r} - \mathbf{r}_j, h\right). \tag{3.4}$$

In SPH a differentiable interpolant of a function can be constructed from its values at the particles by using a differentiable kernel, and thus derivatives of an interpolant can be obtained by standard analytic differentiation. It is therefore a straight forward procedure to define the gradient and the Laplacian of a quantity field, under the assumption that the smoothing kernel is first and second order differentiable, respectively. Considering a system of two particles, i and j, the partial derivative on (3.4) for the x-component becomes

$$\frac{\partial}{\partial x} A_S(\mathbf{r}_i) = \frac{\partial}{\partial x}\left(A_j \frac{m_j}{\rho_j} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right)\right). \tag{3.5}$$

Using the product rule on (3.5) yields

$$\begin{aligned}
\frac{\partial}{\partial x}\left(A_j \frac{m_j}{\rho_j} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right)\right) &= \frac{\partial}{\partial x}\left(A_j \frac{m_j}{\rho_j}\right) W\left(\mathbf{r}_i - \mathbf{r}_j, h\right) + A_j \frac{m_j}{\rho_j} \frac{\partial}{\partial x} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right) \\
&= 0 \cdot W\left(\mathbf{r}_i - \mathbf{r}_j, h\right) + A_j \frac{m_j}{\rho_j} \frac{\partial}{\partial x} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right) \\
&= A_j \frac{m_j}{\rho_j} \frac{\partial}{\partial x} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right),
\end{aligned} \tag{3.6}$$

where we have used the fact that $A_j \frac{m_j}{\rho_j}$ at this point does not directly depends on the x-component, and neither on any other component of space, and hence the product can be considered a constant.

Knowing that the derivatives of a summation interpolant only affect the smoothing kernel, the gradient of the smoothed quantity field (3.4) becomes

$$\nabla A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right). \tag{3.7}$$

To obtain higher accuracy on the gradient of a quantity field, the interpolant can instead be obtained by using

$$\begin{aligned} \rho \nabla A &= \nabla(\rho A) - A\nabla\rho \\ &\Updownarrow \\ \nabla A &= \frac{1}{\rho}\left(\nabla(\rho A) - A\nabla\rho\right), \end{aligned} \tag{3.8}$$

which is an example of the use of the so called second golden rule of SPH, which states that it is better to rewrite formulae with the density placed inside the operators [21]. The first golden rule of SPH is regarding kernels and presented in Section 3.2. The SPH expression for (3.8), using (3.7) on the gradient terms, yields

$$\begin{aligned} \nabla A_S(\mathbf{r}) &= \frac{1}{\rho}\left(\sum_j \rho_j A_j \frac{m_j}{\rho_j} \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right) - A\sum_j \rho_j \frac{m_j}{\rho_j} \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right)\right) \\ &= \frac{1}{\rho}\left(\sum_j A_j m_j \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right) - \sum_j A m_j \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right)\right) \\ &\quad \frac{1}{\rho}\sum_j \left(A_j - A\right) m_j \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right). \end{aligned} \tag{3.9}$$

A particular symmetrized form of (3.8) can be obtained be rewriting $\frac{\nabla A}{\rho}$ according to

$$\begin{aligned} \frac{\nabla A}{\rho} &= \nabla\left(\frac{A}{\rho}\right) + \frac{A}{\rho^2}\nabla\rho \\ &\Updownarrow \\ \nabla A &= \rho\left(\nabla\left(\frac{A}{\rho}\right) + \frac{A}{\rho^2}\nabla\rho\right), \end{aligned} \tag{3.10}$$

which in SPH terms becomes

$$\begin{aligned}\nabla A_S(\mathbf{r}) &= \rho\left(\sum_j \frac{A_j}{\rho_j}\frac{m_j}{\rho_j}\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right)+\frac{A}{\rho^2}\sum_j \rho_j\frac{m_j}{\rho_j}\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right)\right)\\ &= \rho\left(\sum_j \frac{A_j}{\rho_j^2}m_j\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right)+\sum_j \frac{A}{\rho^2}m_j\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right)\right) \qquad (3.11)\\ &= \rho\sum_j\left(\frac{A_j}{\rho_j^2}+\frac{A}{\rho^2}\right)m_j\nabla W\left(\mathbf{r}-\mathbf{r}_j,h\right).\end{aligned}$$

The Laplacian of the smoothed quantity field (3.4) can be derived using the same method as with the gradient, and it becomes

$$\nabla^2 A_S(\mathbf{r}) = \sum_j A_j\frac{m_j}{\rho_j}\nabla^2 W\left(\mathbf{r}-\mathbf{r}_j,h\right). \qquad (3.12)$$

3.2 Smoothing Kernels

The use of different kernels in SPH is analogue to using different difference schemes in finite difference methods, thus the choice of smoothing kernel for a specific problem is of significant importance. The derivatives of the smoothing kernels have an important impact for different SPH estimations, but we will now focus on the kernels and their required properties.

In [21] it is required that a suitable kernel must have the following two properties,

$$\int_\Omega W(\mathbf{r},h)d\mathbf{r} = 1 \qquad (3.13)$$

and

$$\lim_{h\to 0} W(\mathbf{r},h) = \delta(\mathbf{r}), \qquad (3.14)$$

where δ is Dirac's delta function

$$\delta(\mathbf{r}) = \begin{cases}\infty & \|\mathbf{r}\| = 0\\ 0 & \text{otherwise}\end{cases}. \qquad (3.15)$$

Equation (3.13) states that the kernel must be normalized, and that the unit integral ensures that maxima and minima are not enhanced. The kernel must also be positive

$$W(\mathbf{r},h) \geq 0, \qquad (3.16)$$

to ensure that it is an averaging function [7]. If the kernel is even,

$$W(\mathbf{r},h) = W(-\mathbf{r},h)\,, \tag{3.17}$$

then rotational symmetry is enforced, which is useful to ensure invariance under rotations of the coordinate system. If (3.17) and (3.13) are both obtained, i.e. the kernel are even and normalized, then the interpolation is of second order accuracy [21], that is the error of approximating (3.1) by (3.4) is $O\left(h^2\right)$ or better. In [21] it is also suggested that a suitable kernel should have a limited or compact support radius, in order to ensure zero kernel interactions outside the computational range of the radius. We use the kernel width h as the compact support radius for all smoothing kernels, which implies $W(\mathbf{r},h) = 0,\ \|\mathbf{r}\| > h$.

The first golden rule of SPH states that if a new interpretation of an SPH equation is to be found, it is always best to assume the kernel is a Gaussian [21]. The isotropic Gaussian kernel in n dimensions is given by

$$W_{gaussian}(\mathbf{r},h) = \frac{1}{\left(2\pi h^2\right)^{\frac{3}{2}}} e^{-\left(\|\mathbf{r}\|^2/2h^2\right)},\quad h > 0\,, \tag{3.18}$$

and it is depicted on Figure 3.1 in one dimension for $h = 1$.

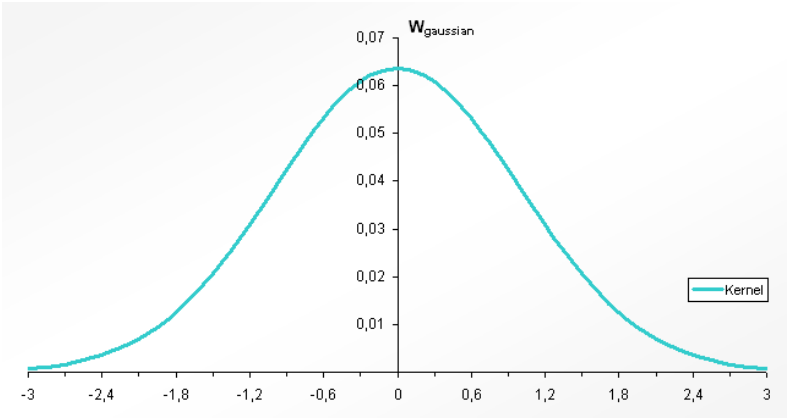


Figure 3.1 The isotropic Gaussian kernel in 1D, for $h = 1$.

Even though a Gaussian kernel has very nice mathematically properties, it is not always the best kernel to use, e.g. it does not have a compact support for our purpose, and it requires the evaluation of the expensive exponential function.

In the next chapter we will focus more intensively at the specific smoothing kernels that are needed for the different purposes within Lagrangian fluid dynamics using smoothed particle hydrodynamics.

3.3 Summery

In this chapter the following important topics are covered/achieved:

- Smoothed particle hydrodynamics is an interpolation method that can approximate continuously field quantities and their derivatives by using discrete sample points, called smoothed particles.
- Particles carry mass, m, position, $\mathbf{r}$, and velocity, $\mathbf{u}$, but can also hold SPH estimated quantities, e.g. mass-density, ρ, pressure, p, etc.
- The following relation between volume, mass, and mass-density applies, and can be used to determined the volume occupied by a particle,
$$V = \frac{m}{\rho}.$$
- The following properties must hold for a smoothing kernel,
$$\int_\Omega W(\mathbf{r},h)d\mathbf{r} = 1 \qquad \text{(normalized)},$$
$$W(\mathbf{r},h) \geq 0 \qquad \text{(positive)},$$
$$W(\mathbf{r},h) = W(-\mathbf{r},h) \qquad \text{(even)}.$$
- We only use smoothing kernels with a compact support radius h, which implies $W(\mathbf{r},h) = 0,\ \|\mathbf{r}\| > h$.
- The basis formulation of SPH to approximate any quantity field and their derivatives yields
$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W\left(\mathbf{r} - \mathbf{r}_j, h\right),$$
$$\nabla A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right),$$
$$\nabla^2 A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W\left(\mathbf{r} - \mathbf{r}_j, h\right).$$
- A symmetrized gradient of a higher accuracy can in SPH be obtained by
$$\nabla A_S(\mathbf{r}) = \rho \sum_j \left(\frac{A_j}{\rho_j^2} + \frac{A}{\rho^2}\right) m_j \nabla W\left(\mathbf{r} - \mathbf{r}_j, h\right).$$
- SPH is originally designed for compressible flow problems.

4 Lagrangian Fluid Dynamics

The equations (2.1) and (2.2) represent the basic Eulerian formulation of an incompressible, isothermal fluid. Using particles instead of a grid simplifies the equations significantly. We assume that the amount of particles is constant during the simulation, and by keeping the mass fixed for each particle, it implies that mass conservation is guaranteed, and that (2.2) can be omitted. Figure 4.1 depicts a basic layout of a particle-based fluid, which has been reduced to two-dimensions for reasons of clarity. In the Lagrangian formulation of a fluid the particles completely define the fluid, which implies that the particles move with the fluid. Compared to the Eulerian view this means that any field quantity now depends on time, t, only. The particles carry mass, position, and velocity, and will hold smoothed quantity approximations obtained from SPH. The acceleration for a Lagrangian fluid particle becomes the ordinary time derivative $\frac{d}{dt}$ of its velocity $\mathbf{u}(t)$. This expounds why the advection term of (2.1) is not present in the Lagrangian view. The basic Lagrangian formulation of the Navier-Stokes equations for an incompressible, isothermal fluid is given by

$$\rho \frac{d\mathbf{u}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}\,. \tag{4.1}$$

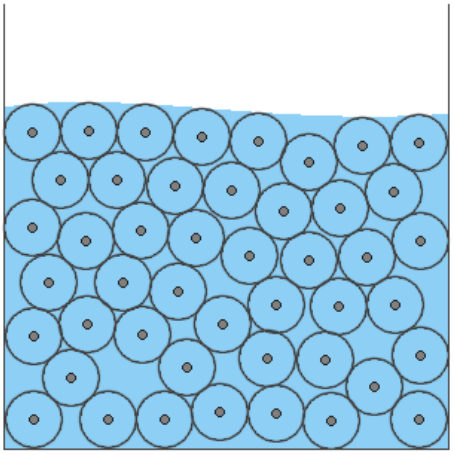


Figure 4.1 Lagrange particle-based fluid structure in 2D. The particles are represented by the dots. The circles represent the volume of each particle.

The right hand side of (4.1) consists of internal and external force fields. The force fields can be combined into a sum of force fields, $\mathbf{F} = \mathbf{f}^{internal} + \mathbf{f}^{external}$. For particle i the acceleration then becomes

$$\mathbf{a}_i = \frac{d\mathbf{u}_i}{dt} = \frac{\mathbf{F}_i}{\rho_i}\,, \tag{4.2}$$

where $\mathbf{a}_i$ and $\mathbf{u}_i$ are the acceleration and velocity of particle i, respectively, $\mathbf{F}_i$ is the total force acting the particle, and ρ_i is the mass-density evaluated at the position of particle i.

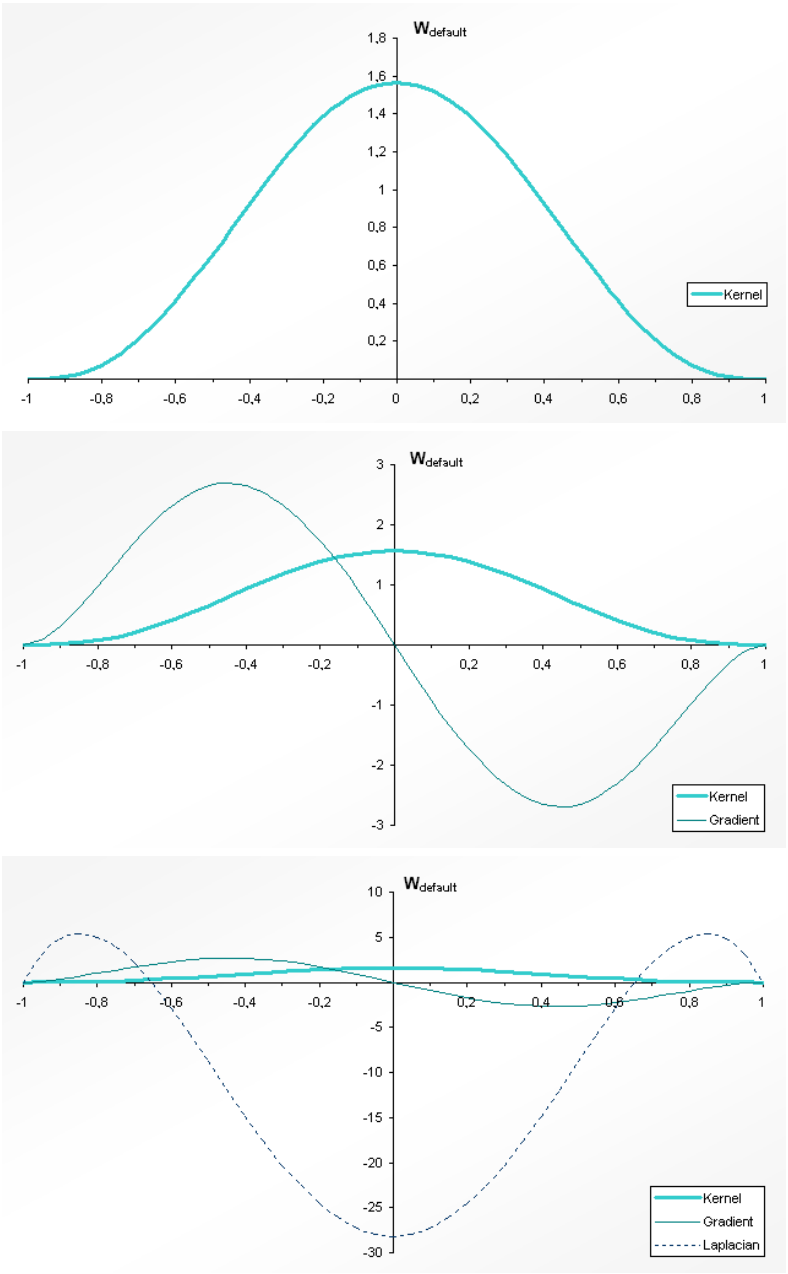


Figure 4.2 The default kernel and its derivatives in one dimension for $h = 1$. For optimal visibility the graphs are scaled differently.

In Section 3.2 we learned about the first golden rule of SPH, but we also concluded that the isotropic Gaussian kernel was not fit to be used for our purpose. We need a default smoothing kernel with compact support for the interparticle-based SPH computations required to solve for (4.1). In [12] several kernels for SPH in stable fields are discussed, and among the kernels with compact support are the B-Spline and Q-Spline kernels, where the Q-Spline is concluded to be the best kernel

in terms of computational accuracy. However, the Q-Spline kernel requires the evaluation of the square root, which can be expensive if the kernel is often used. Instead we will use the 6th degree polynomial kernel suggested by [23] as our default kernel, which is given by

$$W_{default}(\mathbf{r},h) = \frac{315}{64\pi h^9}\begin{cases}\left(h^2-\|\mathbf{r}\|^2\right)^3 & 0\le\|\mathbf{r}\|\le h\\ 0 & \|\mathbf{r}\|>h,\end{cases} \tag{4.3}$$

with the gradient

$$\nabla W_{default}(\mathbf{r},h) = -\frac{945}{32\pi h^9}\mathbf{r}\left(h^2-\|\mathbf{r}\|^2\right)^2, \tag{4.4}$$

and the Laplacian

$$\nabla^2 W_{default}(\mathbf{r},h) = -\frac{945}{32\pi h^9}\left(h^2-\|\mathbf{r}\|^2\right)\left(3h^2-7\|\mathbf{r}\|^2\right). \tag{4.5}$$

Noticeable qualities for the default kernel are that it preserves the Gaussian bell curve and the norm of $\mathbf{r}$ can be omitted completely, thus making it computational pleasing. Figure 4.2 depicts the default kernel and its derivatives in one dimension. The default kernel and its derivatives are used for all smoothed quantity field approximations, except for the internal fluid force fields, see Section 4.2.

4.1 Mass-Density

In Section 3.1 we saw that any continuous quantity field, their gradients, or Laplacians can be approximated by using the SPH formulations (3.4), (3.7), or (3.12), respectively. The equations assume that particle masses and mass-densities for all particles are known prior to applying SPH. The particle mass is a user defined constant, but the mass-density is a continuous field of the fluid, which must be computed. To avoid any contradiction, the mass-density is the one field that alone only depends on the particle mass, and for this specific case we can use (3.4) to compute the field. At particle i the mass-density thus yields

$$\begin{aligned}\rho_i &= \rho(\mathbf{r}_i)\\ &= \sum_j \rho_j \frac{m_j}{\rho_j} W\left(\mathbf{r}_i-\mathbf{r}_j,h\right)\\ &= \sum_j m_j W\left(\mathbf{r}_i-\mathbf{r}_j,h\right).\end{aligned} \tag{4.6}$$

4.2 Internal Forces

Internal force densities are force contributions that only arise from within the fluid, and in (4.1) they are the pressure and viscosity force density fields, which are the first and second term on the right hand side, respectively. The internal fluid force densities all require applying SPH to compute the derivatives of the quantity fields.

4.2.1 Pressure

The pressure p at a particle can be determined using the ideal gas law, which states

$$pV = nRT\,, \tag{4.7}$$

where $V = \frac{1}{\rho}$ is the volume per unit mass, n is the number of gas particles in mol, R is the universal gas constant, and T is the temperature. For an isothermal fluid with a constant mass the right hand side of (4.7) can be kept constant, and hence we replace it by a gas stiffness constant k, which theoretically only depends on the amount of particles in the fluid. The pressure term can then be written as

$$\begin{aligned} pV &= k \\ p\frac{1}{\rho} &= k \\ p &= k\rho\,. \end{aligned} \tag{4.8}$$

If the pressure is known at each particle, the pressure force at particle i, in SPH notation, yields

$$\begin{aligned} \mathbf{f}_i^{pressure} &= -\nabla p(\mathbf{r}_i) \\ &= -\sum_{j\neq i} p_j \frac{m_j}{\rho_j} \nabla W\big(\mathbf{r}_i - \mathbf{r}_j, h\big)\,. \end{aligned} \tag{4.9}$$

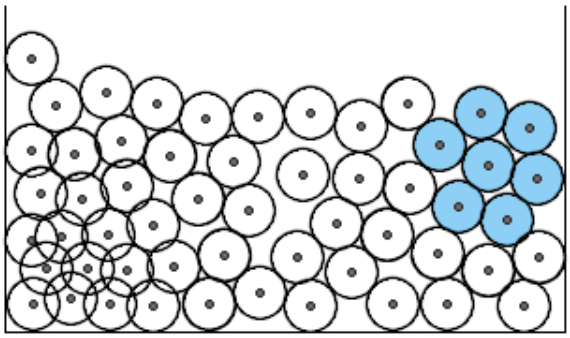
Unfortunately, the pressure force in (4.9) is not symmetrical. This can be verified when only two particles interact, as the first particle only uses the pressure at the second particle to compute its pressure force, and vice versa. Because the pressures at the particles are not equal in general the pressure force will be asymmetric, and the action-reaction law will not be conserved. The SPH formalism has a way of symmetrizing the pressure force. This is complied by employing (3.11) for $-\nabla p(\mathbf{r}_i)$, which yields

$$\mathbf{f}_i^{pressure} = -\rho_i \sum_{j\neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W\big(\mathbf{r}_i - \mathbf{r}_j, h\big). \tag{4.10}$$

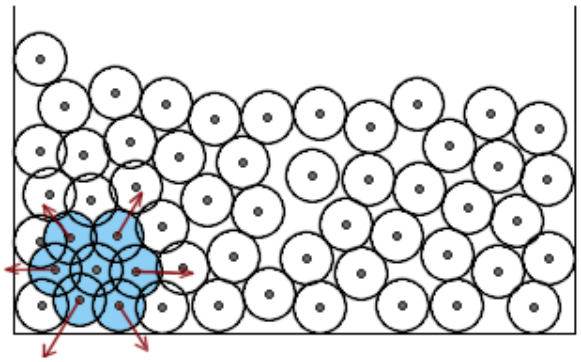
Other solutions to symmetrize asymmetric forces exist. In [23] a simple solution is provided that best suits their purposes of speed and stability,

$$\mathbf{f}_i^{pressure} = -\sum_{j \neq i} \frac{p_i + p_j}{2} \frac{m_j}{\rho_j} \nabla W\left(\mathbf{r}_i - \mathbf{r}_j, h\right), \tag{4.11}$$

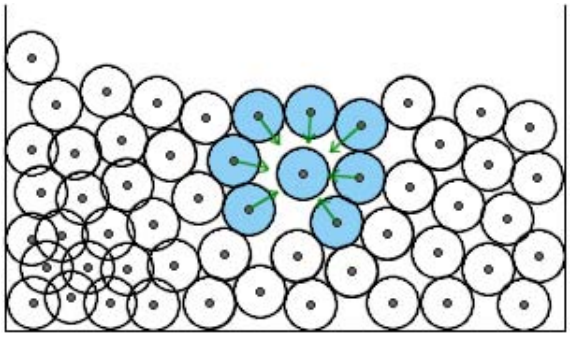
which is symmetric due to it uses the standard arithmetic mean of the pressures of the interacting particles. A symmetrical pressure force has the qualities that linear and angular momentum are conserved, and hence Newton's 3rd law.



a) Balanced mass-density in the marked region, hence no produced pressure forces.



b) High mass-density in the marked region will produce repulsive pressure forces.



c) Low mass-density in the marked region will produce attractive pressure forces.

Figure 4.3 Pressure force behavior from different mass-densities.

Computing the pressure using (4.8) and using the result in the calculation of the pressure force, results in purely repulsive forces between particles, which is true for an ideal gas that tends to expand in space. In contrast, liquids should exhibit

internal cohesion and have a constant mass-density at rest. In [5] Desbrun et al. suggest using a modified version of the ideal gas state equation, with an additional rest pressure p_0, which is

$$\begin{aligned} (p+p_0)V &= k \\ p+k\rho_0 &= k\rho \\ p &= k(\rho-\rho_0), \end{aligned} \tag{4.12}$$

where ρ_0 is the rest density of the fluid. Using (4.12) as the pressure term for the pressure force (4.10) results in an attraction-repulsion force that will be minimized as the density approaches the rest density. Figure 4.3 illustrates the behaviors produced by the pressure force from different mass-densities.

The gradient from the pressure field is used in the calculation of the pressure force. If the gradient of the default kernel (4.4) is used as the choice of smoothing kernel in (4.10), particle clustering will arise in high pressure regions. Particles will build clusters due to $\nabla W_{default}(\mathbf{r},h) \to \mathbf{0}$ as $\|\mathbf{r}\| \to 0$, verifiable on Figure 4.2, which implies that repulsion forces get more attenuated as particles approach each other. We are not interested in clustering caused by an unreliable model of the pressure force. If particle clusters are needed we are interested in an explicit and precise physical description of such a behavior. We need another smoothing kernel for the pressure force. Desbrun et al. became acquainted with the same problem in [5], and proposed another normalized kernel, which later was adopted in [23]. For the evaluations of the pressure force between particles, we employ the spiky kernel from [23] as our pressure kernel, which yields,

$$W_{pressure}(\mathbf{r},h) = \frac{15}{\pi h^6} \begin{cases} (h-\|\mathbf{r}\|)^3 & 0 \le \|\mathbf{r}\| \le h \\ 0 & \|\mathbf{r}\| > h, \end{cases} \tag{4.13}$$

with the gradient

$$\begin{gathered} \nabla W_{pressure}(\mathbf{r},h) = -\frac{45}{\pi h^6}\frac{\mathbf{r}}{\|\mathbf{r}\|}(h-\|\mathbf{r}\|)^2, \\ \lim_{r\to 0^-} \nabla W_{pressure}(r,h) = \frac{45}{\pi h^6}, \quad \lim_{r\to 0^+} \nabla W_{pressure}(r,h) = -\frac{45}{\pi h^6}, \end{gathered} \tag{4.14}$$

and the Laplacian

$$\begin{gathered} \nabla^2 W_{pressure}(\mathbf{r},h) = -\frac{90}{\pi h^6}\frac{1}{\|\mathbf{r}\|}(h-\|\mathbf{r}\|)(h-2\|\mathbf{r}\|), \\ \lim_{r\to 0} \nabla^2 W_{pressure}(r,h) = -\infty, \end{gathered} \tag{4.15}$$

where we have written the limits for one dimension only. Figure 4.4 depicts the pressure kernel and its derivatives in one dimension. Observe that $\nabla W_{pressure}(r,h) \to -\frac{45}{\pi h^6}$ as $r \to 0^+$, which will model the required repulsion force when adjacent particles become too dense.

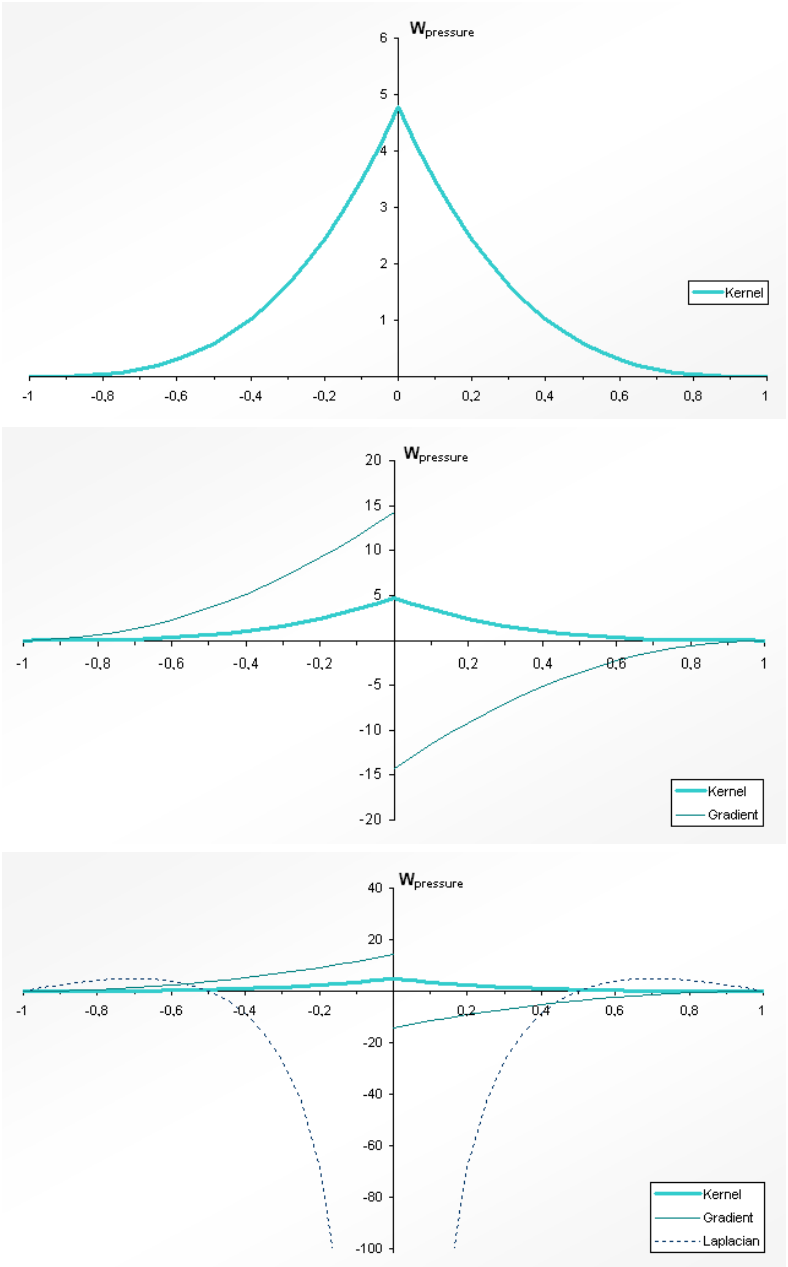


Figure 4.4 The pressure kernel and its derivatives for smoothing radius $h = 1$. For optimal visibility the graphs are scaled differently.

4.2.2 Viscosity

A fluid is a substance that cannot resist shear stress and consequently will flow upon deformation. At the same time when the fluid flows, the molecules undergo internal friction that will decrease its kinetic energy by converting it into heat. The

resistance to flow is called viscosity, and the viscosity coefficient, μ, defines the strength of how viscous the fluid is. The SPH variant to the viscosity force term yields

$$\begin{aligned} \mathbf{f}_i^{viscosity} &= \mu \nabla^2 \mathbf{u}(\mathbf{r}_i) \\ &= \mu \sum_{j \neq i} \mathbf{u}_j \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \end{aligned} \tag{4.16}$$

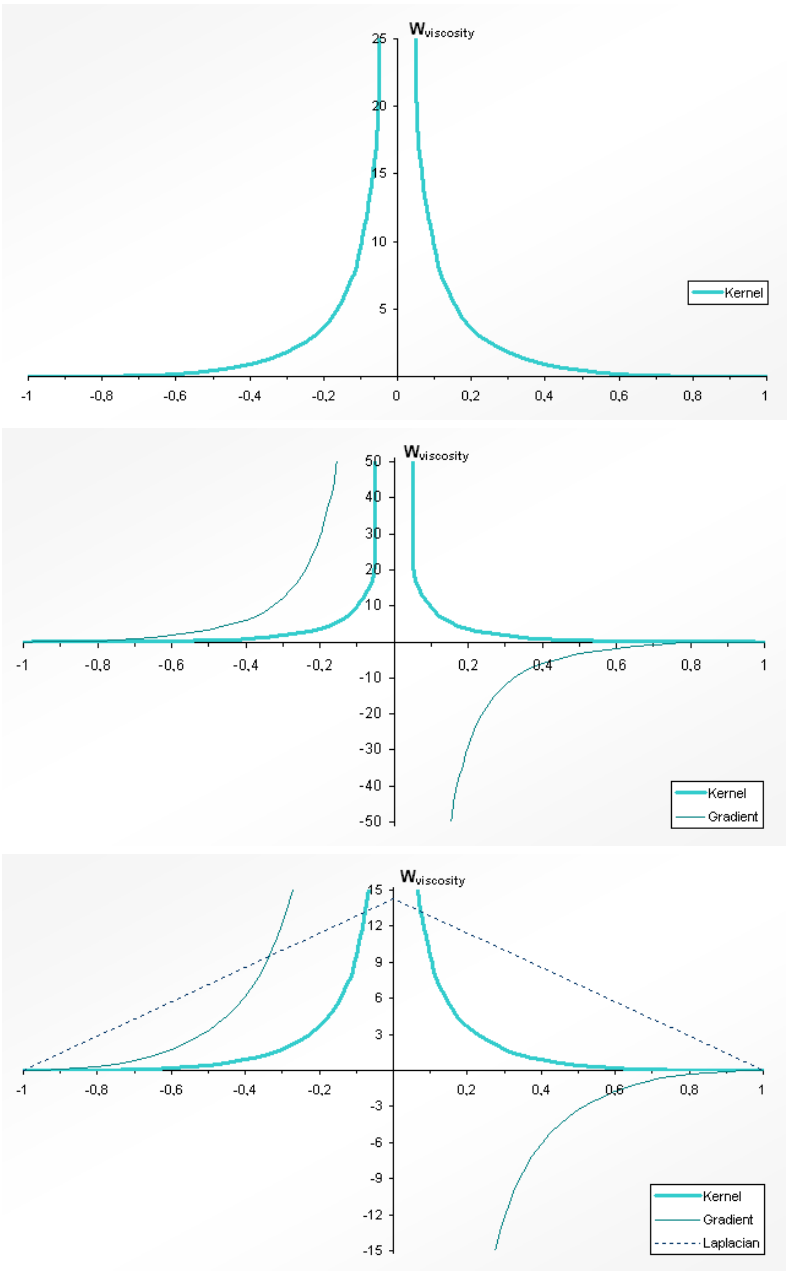


Figure 4.5 The viscosity kernel and its derivatives in one dimension for smoothing length $h = 1$. For optimal visibility the graphs are scaled differently.

Like the pressure force in (4.9) the viscosity force in (4.16) is also asymmetric due to the velocity varies from particle to particle. To counteract this issue Müller et al. in [23] have chosen to symmetrize the velocity fields using

$$\mathbf{f}_i^{viscosity} = \mu \sum_{j \neq i} \big(\mathbf{u}_j - \mathbf{u}_i\big) \frac{m_j}{\rho_j} \nabla^2 W\big(\mathbf{r}_i - \mathbf{r}_j, h\big), \tag{4.17}$$

which is possible due to the fact that viscosity forces only depend on velocity differences and not on absolute velocities. To justify (4.17) we can employ the second golden rule of SPH that is, to write the viscosity term with the density placed inside the Laplacian operator,

$$\mu \nabla^2 \mathbf{u} \;=\; \frac{\mu}{\rho}\big(\nabla^2 (\rho \mathbf{u}) - \mathbf{u} \nabla^2 \rho\big), \tag{4.18}$$

which in SPH formulation, using the structure of (3.9), becomes

$$\mathbf{f}_i^{viscosity} = \frac{\mu}{\rho_i} \sum_j \big(\mathbf{u}_j - \mathbf{u}_i\big) m_j \nabla^2 W\big(\mathbf{r}_i - \mathbf{r}_j, h\big). \tag{4.19}$$

Equation (4.17) can be written as (4.19) under the assumption that the mass-density is constant and identical at all particles. The mass-density is computed using (4.6), thus it generally varies from particle to particle, and consequently why (4.17) seems more correct to use than (4.19).

The Laplacian of the smoothing kernel in (4.17) is constrained to be positive. This is required because we do not want the forces due to viscosity to increase the relative velocity, and thereby introduce energy and instability into the system. If the Laplacian is positive everywhere only then will the viscosity force work as a damping term, and damp the relative velocity. The standard kernel does not have this property and neither does the pressure kernel, as can be verified on Figure 4.4, thus we need a smoothing kernel with $\nabla^2 W(\mathbf{r}, h) \geq 0$ for $\|\mathbf{r}\| \leq h$. We employ the viscosity kernel proposed by [23], which yields

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

$$\lim_{r \to 0} W_{viscosity}(r, h) \;=\; \infty, \tag{4.20}$$

with the gradient

$$\nabla W_{viscosity}(\mathbf{r},h) = \frac{15}{2\pi h^3}\mathbf{r}\left(-\frac{3\|\mathbf{r}\|}{2h^3}+\frac{2}{h^2}-\frac{h}{2\|\mathbf{r}\|^3}\right),$$
$$\lim_{r\to 0^-}\nabla W_{viscosity}(r,h) = +\infty, \quad \lim_{r\to 0^+}\nabla W_{viscosity}(r,h) = -\infty, \tag{4.21}$$

and the Laplacian

$$\nabla^2 W_{viscosity}(\mathbf{r},h) = \frac{45}{\pi h^6}(h-\|\mathbf{r}\|), \tag{4.22}$$

where we have written the limits for one dimension only. Figure 4.5 depicts the viscosity kernel and its derivatives in one dimension for smoothing radius $h = 1$.

4.3 External Forces

External force densities are balanced against the internal force densities. In (4.1) the external force density field is the last term on the right hand side, which can be combined into a sum of force densities,

$$\mathbf{f}^{external} = \sum_n \mathbf{f}^n, \tag{4.23}$$

where n indicates each individual external force density. Some external force contributions can be applied directly to the particles without the use of SPH, while others still depend on adjacent particles. Collision handling, which also can be considered as an external force contribution, is covered in Section 4.4.

4.3.1 Gravity

The gravitational force density field is acting equally on all fluid particles, and is given by

$$\mathbf{f}_i^{gravity} = \rho_i \mathbf{g}, \tag{4.24}$$

where $\mathbf{g}$ is the downward gravitational acceleration.

4.3.2 Buoyancy

For gaseous fluids we want the particles to buoyant. Buoyancy is caused by diffusion of temperatures, but as we are modeling an isothermal fluid, an artificial buoyancy force density can be employed as

$$\mathbf{f}_i^{buoyancy} = b(\rho_i - \rho_0)\mathbf{g}, \tag{4.25}$$

where $b > 0$ is the artificial buoyancy diffusion coefficient. The buoyancy force will make particles buoyant if the mass-density has become smaller than the desired rest density. For a gas we will only employ the buoyancy force in preference to gravity.

4.3.3 Surface Tension

The surface tension force is an external force density that can be applied to the free surface of a liquid fluid. It is normally not a part of the Navier-Stokes equations, due to it is considered a boundary condition. For a Lagrangian fluid the boundaries can be identified by the particles. Fluid molecules are influenced by attractive forces from adjacent molecules, and these forces are kept in perfect balance inside the fluid. On the fluid surface the forces are unbalanced and cause surface tensions. The surface tension forces act in the direction of the inward surface normals towards the fluid, where they bind the fluid surface together. The surface tension force will flatten the surface curvature by minimizing the surface area. Figure 4.6 depicts the behavior of the surface tension force.

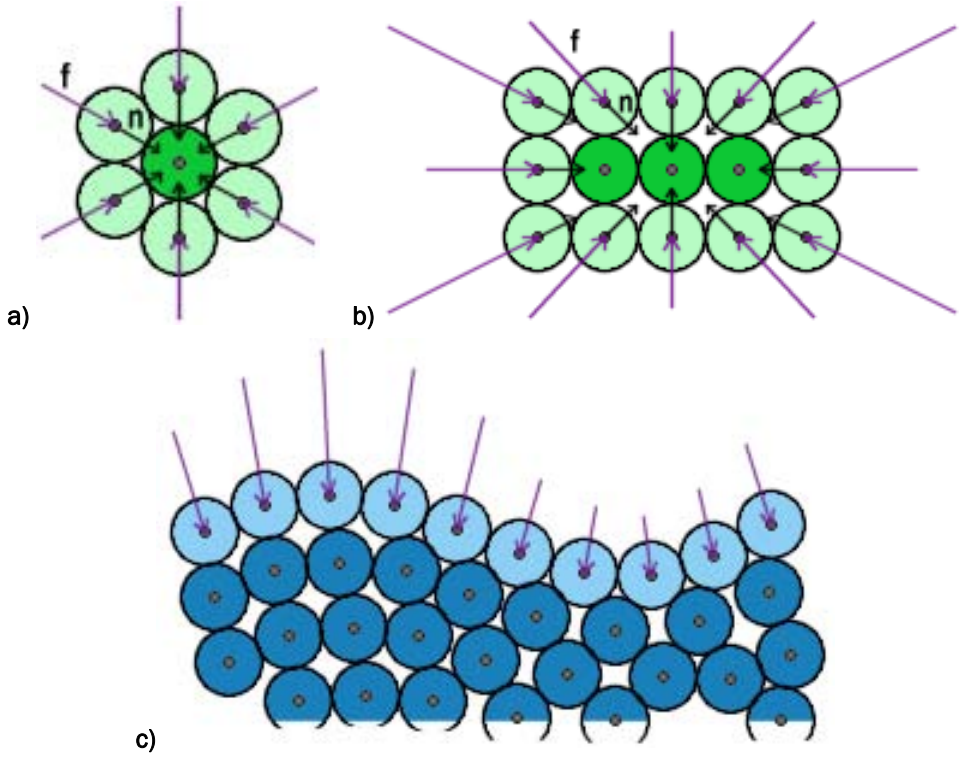


Figure 4.6 Behavior of the surface tension force. Inward surface normals, $\mathbf{n}$, point from the particles towards the fluid. Surface tension forces, $\mathbf{f}$, end at the particles. **a)** The surface tension force acts with the same strength on a spherical shape. **b)** The strengths of the surface tension forces depend on the curvature. **c)** A positive curvature (left) generates a stronger surface tension force than a negative curvature (right), but the surface tension force will in all cases work in the direction towards the fluid.

The surface tension force we employ is explicitly based on the surface tension model from [23], which yields

$$\mathbf{f}_i^{surface} = -\sigma\nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|}, \tag{4.26}$$

where $\mathbf{n}_i$ is the inward surface normal of the fluid at particle i, c_i is the smoothed value of the color field evaluated at particle i, and σ is the tension coefficient that depends on the fluids that form the surface, e.g. water and air.

The color field c is an additional field quantity with $c=1$ exactly at particle locations and $c=0$ everywhere else. In SPH formulation the smoothed color field at particle i is

$$\begin{aligned} c_i &= c(\mathbf{r}_i) \\ &= \sum_j c_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h) \\ &= \sum_j \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h). \end{aligned} \tag{4.27}$$

The gradient of the smoothed color field yields the inward surface normal of the fluid,

$$\begin{aligned} \mathbf{n}_i &= \nabla c(\mathbf{r}_i) \\ &= \sum_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h), \end{aligned} \tag{4.28}$$

where $\|\mathbf{n}_i\| > 0$ only near and on the surface of the fluid. The divergence of $\mathbf{n}$ measures the Gaussian curvature of the surface,

$$\kappa = -\frac{\nabla \mathbf{n}}{\|\mathbf{n}\|} = -\frac{\nabla^2 c}{\|\mathbf{n}\|}, \tag{4.29}$$

where the negation is necessary to get a positive curvature for convex fluid volumes.

The surface traction, or external traction as opposed to internal traction that concerns stress fields, is force per unit area acting on a given location on the fluid's surface. In [23] the surface traction is defined as

$$\mathbf{t} = \sigma\kappa \frac{\mathbf{n}}{\|\mathbf{n}\|}, \tag{4.30}$$

and it should only be distributed to particles near and on the surface. As $\|\mathbf{n}_i\|$ gets smaller when particle i is away from the surface, we can multiply the surface

traction by a normalized scalar field $\delta_i = \|\mathbf{n}_i\|$. This will make sure that the force density is spread onto all potential particles, and ultimately results in the surface tension force,

$$\mathbf{f}_i^{surface} = \delta_i \mathbf{t}_i = \sigma \kappa_i \mathbf{n}_i = -\sigma \nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|}. \tag{4.31}$$

The surface tension force only applies to particles located near or on the liquid surface. This constraint is determined numerically as $\frac{\mathbf{n}}{\|\mathbf{n}\|}$ becomes numerical unstable when $\|\mathbf{n}\| \to 0$. One way to prevent numerical problems when evaluating (4.31) for particle i, is to compute $\mathbf{f}_i^{surface}$ only when

$$\|\mathbf{n}_i\| \geq \ell, \tag{4.32}$$

where $\ell > 0$ is some threshold relating to the particle concentration.

The surface tension force is asymmetric by design. In the real world surface tension forces are the consequence of fluid-fluid interactions, e.g. between water and air molecules. We only model a tiny fraction of the real world, and do not have any air particles to symmetrize the surface tension force, thus the model might be in absence of some realism.

4.3.4 User Interaction

A part of the definition of interactive simulation states that an end user should be able to interact with the fluids. This can be realized by controlling the different internally and externally physical properties of the fluids during the simulation, e.g. the viscosity strengths, the rest densities, and the gravity. Very often interactivity is introduced by letting the end user control the transformation of collision containers and obstacles, and thus implicitly affect the fluids by having an effect on the environment, e.g. throwing rigid items into a fluid, or moving around a glass of water. External force density fields can also be introduced and removed dynamically at runtime to model environmental effects, e.g. a rotational swirl or a blowing wind.

4.4 Collision Handling

The small-scale working domain of interactive Lagrangian fluids is limited. A practical way of meeting a convincing environment of the fluid is to constraint the particle system within well defined boundaries. Boundary containers, such as boxes, spheres, and capsules, are commonly used to constraint a fluid. When particles collide with a container they must stay inside its boundaries. Likewise, if particles collide with an obstacle, they may not penetrate or gain access to the interior of the object.

Collision handling can be divided into two sub parts; collision detection and collision response. In this section collision detection between fluid particles and their surroundings will be treated. For the fluid's point of view all obstacles and containers are assumed to be fixed and rigid, hence only collision responses to fluid particles will be considered. In the real world when a fluid collides with its surroundings, it also inflicts a reaction. This is in agreement with Newton's 3rd law, and should be considered in the case that obstacles can be moved freely according to rigid body mechanics, or the surroundings are deformable.

Containers and obstacles can be represented geometrically or analytically. For the geometrical representation tetrahedra meshes are often applied, as any arbitrary model can be handled alike. The analytical representation is often restricted to standard implicit primitives, as one such primitive can be described by a single mathematical equation that only has a small computational cost. We want to concentrate on finding a good collision handling method for Lagrangian fluids. To omit unnecessary challenges that can arise from the tetrahedra meshes, we will use implicit primitives as the collision handling prototype. Once a stable collision handling has been achieved for implicit primitives, it should be possible to extend the collision objects to tetrahedra meshes. In Section 4.4.4 we will discuss some of the considerations that are necessary for the collision detection of tetrahedra meshes.

4.4.1 Collision Detection

Particles carry position and velocity, which are enough to determine any collision. If the current particle position does not provide enough information to secure a valid impact, the velocity can be used to trace the particle back to its previous position. A generic collision detection system must detect impacts and penetrations, and collect enough information on the collisions to handle collision responses satisfactory. Mandatory information from a collision includes:

- Contact point / Point of impact on the surface, $\mathbf{cp}$.
- Penetration depth through the obstacle, d.
- Surface normal at the contact point, $\mathbf{n}$.

A contact point is where a penetration between a particle and an object has occurred, i.e. the point of impact. The distance the particle has traveled inside the collision obstacle, or outside the collision container, is the penetration depth. The surface normal is a vector at the contact point with direction away from the object. For the rest of this section we will assume the surface normal is the unit vector.

We only consider collisions where the penetration depth is positive, $d > 0$, hence if a particle is located on the surface of an object, it is not colliding. For an implicit primitive it is a straight forward procedure to determine if a particle has penetrated the surface, see next Subsection 4.4.2. However, it is not always straight forward to determine the actual contact point. Consider Figure 4.7, which illustrates a 2D version of a typical problem regarding the collision determination between a particle

and an implicit primitive. The correct contact point is $\mathbf{cp}_1$, with penetration depth d_1, and contact normal $\mathbf{n}_1$, but finding $\mathbf{cp}_1$ is normally not cheap, as it includes line intersection checks against the implicit primitives. The discussions in Subsection 4.4.4 address the similar problem for tetrahedra meshes. However, the contact point $\mathbf{cp}_2$ and its accompanying information are easily retrieved for implicit primitives. In practice, using contact point $\mathbf{cp}_2$ instead of $\mathbf{cp}_1$ is of no significant importance. We will argue that the situation illustrated on Figure 4.7 only occurs if we are using a large integration time step, or very small collision objects. See Section 4.5 and Subsection 5.3.3 regarding the time integration and the time step, respectively. Often the difference between the correct contact point $\mathbf{cp}_1$ and the approximating contact point $\mathbf{cp}_2$ is very small, and the collision response to $\mathbf{cp}_1$ will most likely advance the particle towards $\mathbf{cp}_2$ at a later time.

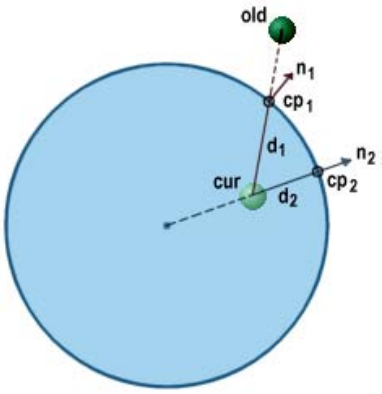


Figure 4.7 Two possible collision determinations from the same particle position update.

4.4.2 Implicit Primitives

Implicit primitives are easy to work with and provide a great deal of information with little computational cost. Verifying particle penetrations against implicit primitives only require the current particle positions. An implicit given two-manifold is described by a function $F : \mathbb{R}^3 \to \mathbb{R}$, for which the surface is determined implicitly by $\left\{\mathbf{x} \in \mathbb{R}^3 \mid F(\mathbf{x}) = 0\right\}$. For the rest of this section we will assume the implicit two-manifolds are closed, i.e. they can be considered as “watertight” solids, with the following conversions,

$$\begin{aligned} F(\mathbf{x}) < 0 \quad & \mathbf{x} \text{ is inside the primitive,} \\ F(\mathbf{x}) = 0 \quad & \mathbf{x} \text{ is on the surface,} \\ F(\mathbf{x}) > 0 \quad & \mathbf{x} \text{ is outside the primitive.} \end{aligned} \tag{4.33}$$

We are using three types of implicit primitives; spheres, capsules, and boxes. Definitions of the primitives and details on how to compute the mandatory collision information are discussed in the following. We are making use of the signum function, which returns the sign of a real number. The signum function is defined by

$$\operatorname{sgn}(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}. \tag{4.34}$$

4.4.2.1 Spheres

Intersection tests between a sphere and other primitives can be performed very efficiently, thus a sphere is a very common bounding volume and collision primitive in Computer Graphics. A sphere is defined implicitly by

$$F_{sphere}(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}\|^2 - r^2, \tag{4.35}$$

where $\mathbf{c}$ and r are the center and radius of the sphere, respectively.

A sphere can be used as either a container or as an obstacle. If a collision has occurred, the contact point is

$$\mathbf{cp}_{sphere} = \mathbf{c} + r\frac{\mathbf{x} - \mathbf{c}}{\|\mathbf{x} - \mathbf{c}\|}, \tag{4.36}$$

the penetration depth is

$$d_{sphere} = |\|\mathbf{c} - \mathbf{x}\| - r|, \tag{4.37}$$

and the unit surface normal is

$$\mathbf{n}_{sphere} = \operatorname{sgn}\left(F_{sphere}(\mathbf{x})\right)\frac{\mathbf{c} - \mathbf{x}}{\|\mathbf{c} - \mathbf{x}\|}. \tag{4.38}$$

4.4.2.2 Capsules

A capsule is a very flexible collision object, and it is used throughout the graphics industry, e.g. to represent various limbs for ragdolls in computer games. Geometrically it can be constructed from a cylinder and two hemispheres, see Figure 4.8.

Implicit primitives can be combined to form more complex types. Constructive solid geometry (CSG) for implicit functions is a simple operation of taking the minima of all individual implicit primitives, e.g.

$$F_{capsule}(\mathbf{x}) = \min\left\{F_{sphere1}(\mathbf{x}), F_{cylinder}(\mathbf{x}), F_{sphere2}(\mathbf{x})\right\}, \tag{4.39}$$

but a capsule can be described implicitly even more beautifully. The idea is to ignore the geometrically primitives and think of a capsule as a fixed distance from a line segment, see Figure 4.9 for the 2D equivalent.

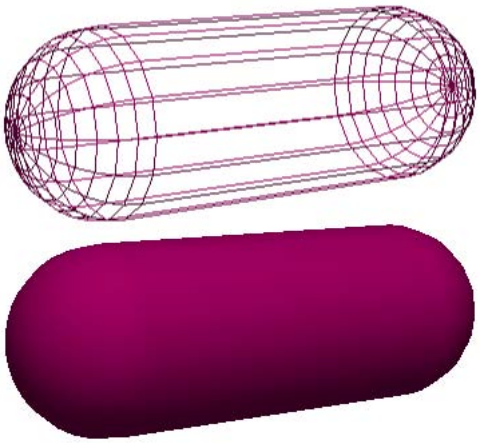


Figure 4.8 A capsule in wireframe (upper) and in solid (lower).

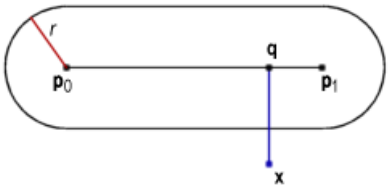


Figure 4.9 A capsule defined by the two end points $\mathbf{p}_0$ and $\mathbf{p}_1$, and a radius r.

If a capsule is defined by two end points $\mathbf{p}_0$ and $\mathbf{p}_1$, and a radius r, we define the implicit function as

$$F_{capsule}(\mathbf{x}) = \|\mathbf{q} - \mathbf{x}\| - r\,, \tag{4.40}$$

where

$$\mathbf{q} = \mathbf{p}_0 + \left(\min\left(1, \max\left(0, -\frac{(\mathbf{p}_0 - \mathbf{x})\cdot(\mathbf{p}_1 - \mathbf{p}_0)}{\|\mathbf{p}_1 - \mathbf{p}_0\|^2}\right)\right)\right)(\mathbf{p}_1 - \mathbf{p}_0)\,. \tag{4.41}$$

Going into details on (4.41), $\mathbf{q} = \mathbf{l}(t)$ is a point on the line segment from $\mathbf{p}_0$ to $\mathbf{p}_1$,

$$\mathbf{l}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0) \quad , \quad 0 \le t \le 1\,. \tag{4.42}$$

The first task is to find t such that $(\mathbf{q} - \mathbf{x})$ is perpendicular to $(\mathbf{p}_1 - \mathbf{p}_0)$, i.e. $\|\mathbf{q} - \mathbf{x}\|$ is the shortest distance from $\mathbf{x}$ to $(\mathbf{p}_1 - \mathbf{p}_0)$,

$$\begin{aligned} 0 &= (\mathbf{q}-\mathbf{x})\cdot(\mathbf{p}_1-\mathbf{p}_0) \\ 0 &= \big(\mathbf{p}_0+t(\mathbf{p}_1-\mathbf{p}_0)-\mathbf{x}\big)\cdot(\mathbf{p}_1-\mathbf{p}_0) \\ t &= -\frac{(\mathbf{p}_0-\mathbf{x})\cdot(\mathbf{p}_1-\mathbf{p}_0)}{\|\mathbf{p}_1-\mathbf{p}_0\|^2} \end{aligned} \tag{4.43}$$

Obtaining a value for t using (4.43) guarantees that $(\mathbf{l}(t)-\mathbf{x})$ is perpendicular to $(\mathbf{p}_1-\mathbf{p}_0)$, but $0\leq t\leq 1$ is not guaranteed. To satisfy (4.42) t is clamped using

$$t=\min\big(1,\max(0,t)\big), \tag{4.44}$$

and $\mathbf{q}$ will now lie on the line segment $(\mathbf{p}_1-\mathbf{p}_0)$.

Dependent on the capsule is used as a collision container or obstacle then (4.33) can be used to determine whether a particle penetration has occurred. The contact point is always

$$\mathbf{cp}_{capsule}=\mathbf{q}+r\frac{\mathbf{x}-\mathbf{q}}{\|\mathbf{x}-\mathbf{q}\|}, \tag{4.45}$$

the penetration depth is

$$d_{capsule}=\big|F_{capsule}(\mathbf{x})\big|, \tag{4.46}$$

while the unit surface normal is

$$\mathbf{n}_{capsule}=\operatorname{sgn}\big(F_{capsule}(\mathbf{x})\big)\frac{\mathbf{q}-\mathbf{x}}{\|\mathbf{q}-\mathbf{x}\|}. \tag{4.47}$$

4.4.2.3 Boxes

An oriented bounding box (OBB) is not geometrically smooth like the sphere or capsule, and cannot be defined as a continuous function. However, respecting the conversions from (4.33), we describe an OBB implicitly by

$$F_{box}(\mathbf{x})=\big[|\mathbf{x}_{local}|-\mathbf{ext}\big]_{\max}, \tag{4.48}$$

where $[\cdot]_{\max}$ is the operator that returns the vector component with the largest value and $\mathbf{ext}$ is the axis extends from the center of the OBB. $\mathbf{x}_{local}$ is $\mathbf{x}$ expressed in body frame (BF) and is defined as a standard transformation from a global to a local frame,

$$\mathbf{x}_{local} = \mathbf{R}^T (\mathbf{x} - \mathbf{c}) , \tag{4.49}$$

where $\mathbf{R}$ and $\mathbf{c}$ are the axis orientation and the center of the OBB, respectively, in the world coordinate system (WCS).

A box can also be used as an obstacle and a container. We are only defining the collision information when the implicit OBB is used as a container, but a similar procedure can be generated for an OBB as an obstacle. When a particle with position $\mathbf{x}$ has penetrated the OBB, use (4.49) to transform the point into $\mathbf{x}_{local}$. The local contact point is computed by

$$\mathbf{cp}_{local} = \min\left[\mathbf{ext}, \max\left[-\mathbf{ext}, \mathbf{x}_{local}\right]\right], \tag{4.50}$$

but we need the contact point in WCS, and transform it back using

$$\mathbf{cp}_{box} = \mathbf{c} + \mathbf{R}\,\mathbf{cp}_{local} . \tag{4.51}$$

The penetration depth is simply the length between the penetrating point and the contact point,

$$d_{box} = \left\|\mathbf{cp}_{box} - \mathbf{x}\right\|, \tag{4.52}$$

and the unit surface normal can be computed by

$$\mathbf{n}_{box} = \frac{\mathbf{R}\,\mathrm{sgn}\left[\mathbf{cp}_{local} - \mathbf{x}_{local}\right]}{\left\|\mathbf{R}\,\mathrm{sgn}\left[\mathbf{cp}_{local} - \mathbf{x}_{local}\right]\right\|}, \tag{4.53}$$

where $\mathrm{sgn}[\cdot]$ is the vector signum operator, e.g. it returns a new vector using (4.34) on each component.

4.4.3 Collision Response

The field of collision response focuses on how to handle the information retrieved from the collision detection satisfactory. Several methods to respond to a collision have been employed in Computer Graphics, some of which will be surveyed briefly and categorized according to our own interpretation.

Acceleration-based collision responses are due to applied external forces, such as spring forces. In [22] spring forces with different spring constants are used to control particles that approach collision obstacles. Penalty forces are exponential spring forces that are applied to particles when they get too close to, or penetrate, collision objects [35]. Collision objects are surrounded by continuous, potential energy functions that generate repulsive forces, thus collision detection is implicitly

applied in the energy potential. However, spring forces cannot always guarantee no penetrations will occur, whereas penalty forces often react too violently, which course instabilities. Our category for the acceleration-based collision responses should not be confused by terms from rigid body dynamics.

Impulse-based collision responses are event driven methods that can be employed to avoid penetrations. An impulse is defined as the time integral of force, and involves modifying the particle velocity at the exact time of collision. An impulse-based collision response is often applied in rigid body dynamics [7].

Collision responses by projections are easy ways to handle particle penetrations. Particles are simply projected out from penetrated objects. A projection of a particle can affect the energy conservation, as the projection can cause an increase in potential energy that in time converts into kinetic energy.

4.4.3.1 Hybrid Impulse-Projection method

The method we employ to solve our fluid particle collisions is a standard explicit project-and-reflect method. If particle i has penetrated an implicit primitive, its position $\mathbf{r}_i$ is modified by projecting the particle back onto the surface along the surface normal $\mathbf{n}$, with a magnitude proportional to the penetration depth d,

$$\mathbf{r}_i = \mathbf{r}_i + d\mathbf{n}\,, \tag{4.54}$$

which for an implicit primitive equals the contact point $\mathbf{cp}$,

$$\mathbf{r}_i = \mathbf{cp}\,. \tag{4.55}$$

The particle velocity $\mathbf{u}_i$ is reflected in the surface normal using the standard vector reflection method

$$\mathbf{u}_i = \mathbf{u}_i - 2(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n}\,, \tag{4.56}$$

but this will result in a perfect elastic collision, i.e. the kinetic energy is conserved. Generally, we believe that fluids do not bounce back upon collisions, thus we do not want to only have elastic collisions. We need to control how much of the kinetic energy that is conserved after a collision, and thus introducing the restitution into (4.56), which yields

$$\mathbf{u}_i = \mathbf{u}_i - (1 + c_R)(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n}\,, \tag{4.57}$$

where $0 \leq c_R \leq 1$ is the coefficient of restitution. To cancel out the velocity in the normal direction we set $c_R = 0$, which models the normally applied no-slip condition

for a liquid, i.e. an inelastic collision, while $c_R = 1$ models an elastic collision as in (4.56).

The position is projected back onto the collision surface, thus we seek to model an impulse-based collision at the exact time the collision occurred. Dependent on the magnitude of c_R at the time of collision, equation (4.57) can wrongfully increase the particle's kinetic energy. To constraint that the outgoing energy must never exceed the incoming energy, we only want to reflect the velocity that was omitted in the collision. Introducing the ratio of the penetration depth to the distance between the last particle position and the penetrating position, yields

$$\mathbf{u}_i = \mathbf{u}_i - \left(1 + c_R \frac{d}{\Delta t \|\mathbf{u}_i\|}\right)(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n}\,, \tag{4.58}$$

where we have implicitly assume that a collision has occurred, i.e. $\|\mathbf{u}_i\| > 0$.

4.4.4 Discussion

Implicit primitives are great for collision prototyping, and provide a computationally inexpensive solution for a variety of collision objects that are regularly used in Computer Graphics. If more complex objects are required, implicit primitives can be combined using CSG methods, but the complexity of the computations will rise accordingly. Tetrahedra meshes are volumetrically by definition, and can be employed as collision objects exactly alike, meaning there are no difference between collision containers and obstacles. Figure 4.10 depicts an example of a capsule shell mesh built from tetrahedra.

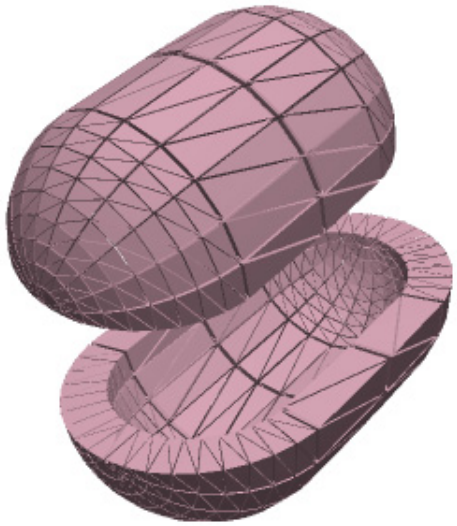


Figure 4.10 A capsule shell built from tetrahedra. The capsule is cut in half and the top has been lifted to make the intrinsic hull visible.

Generic collision detection for tetrahedra meshes includes a detection of points being inside a tetrahedron. For this purpose we convert a point into Barycentric

coordinates $(w_1, w_2, w_3, w_4)^T$ with respect to the tetrahedron. Please consult [7] for details on how to compute the Barycentric coordinates. The point lies inside the tetrahedron if

$$w_1 \geq 0 \wedge w_2 \geq 0 \wedge w_3 \geq 0 \wedge w_4 \geq 0$$

and (4.59)

$$w_4 \leq 1 - (w_1 + w_2 + w_3).$$

Please notice that if one, two, or three of the Barycentric coordinates are zero, the point is located on a triangle, edge, or vertex, respectively, and thus cannot be defined as being completely inside the tetrahedron. The small computational cost of converting a point into Barycentric coordinates, to verify whether or not the point is inside the tetrahedron, is well spent compared to the computations needed for other inside checks, such as oriented half-space plane checks, etc.

Once we have determined if a particle penetrates a tetrahedron, we further need to determine the collision information listed in Subsection 4.4.1. To do this, the four triangles from the tetrahedron are used. The Barycentric coordinates can also be used to determine which triangle of the tetrahedron is the closest one to the penetrating particle, but this does not guarantee in the correct collision information, as the particle can be arbitrary close to any of the three wrong triangles. Instead, we can use the particle velocity to find its previous position, and now find the correct penetration triangle using line-triangle intersection tests, see Figure 4.11**a)** for a 2D equivalent.

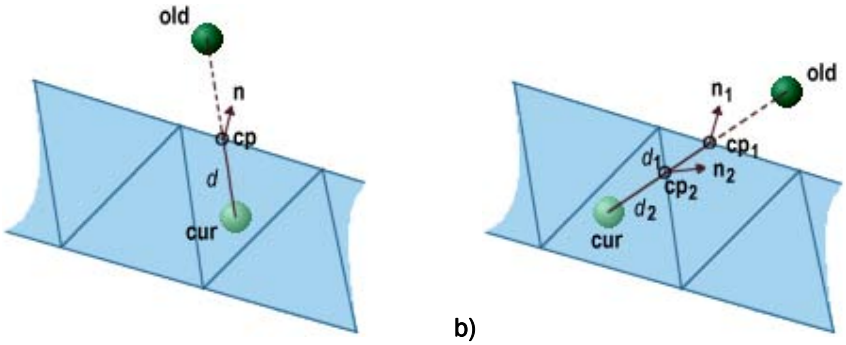


Figure 4.11 Particle colliding against a triangle mesh. **a)** The penetrating edge from the penetration triangle is used for the collision information. **b)** Only to consider the edges from the penetration triangle is fatal, because the particle still will be penetrating the mesh after the collision response.

One disadvantage of using point-tetrahedron collision checks is the possibility that particles can jump right through tetrahedra without being detected, due to too small tetrahedra or too large particle movement. Equivalently, neither can we be sure that any triangle from the detected tetrahedron is the correct one to use for the collision information determination, which can be verified on Figure 4.11**b)**. To only consider the triangles from the tetrahedron that the particle is penetrating, will be fatal for the simulation, because the particle still will be penetrating the collision

object after the collision response. The solution is to detect all tetrahedra and their triangles penetrated by the particle trajectory. The triangle which is closest to the previous particle position must always be used to compute the collision information.

Tetrahedra and their vertices can be arranged into a spatial partition structure for fast retrieval based on query points and lines, see Section 5.1 for an identical method for efficiently retrieval of particle neighborhoods.

4.5 Numerical Time Integration

To simulate the fluid flow, each particle is advanced through time using a global fixed time step Δt. Equation (4.2) is employed to compute the particle acceleration, and the new particle position is obtained from integrating the acceleration numerically. In this section three different integration schemes will be described briefly.

4.5.1 The Implicit Euler Scheme

The Implicit Euler scheme is actually a semi-implicit method, as it is only the position update that is implicit. Semi-implicit Euler is based on the explicit Euler scheme, which probably is the most common integration method. In explicit Euler the position and velocity are updated in parallel,

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{u}_t \,, \tag{4.60}$$

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta t \mathbf{a}_t \,. \tag{4.61}$$

The semi-implicit Euler is no longer independent of the position and velocity updates. The velocity update is the same as (4.61), but the position update uses the result from the velocity update to predict the new position,

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{u}_{t+\Delta t} \,. \tag{4.62}$$

4.5.2 The Verlet Scheme

The Verlet integration method [37] originates from molecular dynamics, and is based on implicit Euler. The variant presented in this subsection is based upon [15]. The current velocity can be estimated using the forward first-order difference operator on positions given by

$$\mathbf{u}_t \approx \frac{\mathbf{r}_t - \mathbf{r}_{t-\Delta t}}{\Delta t} \,. \tag{4.63}$$

Inserting (4.61) into (4.62), and using (4.63) as the current velocity, the new position can be determined as

$$\begin{aligned} \mathbf{r}_{t+\Delta t} &= \mathbf{r}_t + \Delta t\left(\mathbf{u}_t + \Delta t\mathbf{a}_t\right) \\ &= \mathbf{r}_t + \Delta t\left(\left(\frac{\mathbf{r}_t - \mathbf{r}_{t-\Delta t}}{\Delta t}\right) + \Delta t\mathbf{a}_t\right) \\ &= 2\mathbf{r}_t - \mathbf{r}_{t-\Delta t} + \Delta t^2\mathbf{a}_t. \end{aligned} \tag{4.64}$$

The Verlet scheme is one of the computationally fastest integrators and it is usually very stable, as the velocity is given implicitly and will not get out of sync with the position. However, collision responses are not trivial to handle, as it includes modifying positions rather than velocities.

4.5.3 The Leap-Frog Scheme

The leap-frog integration [6] has got its name from the fact that the velocities leap over the positions, and vice versa. Figure 4.12 illustrates the concept. The integration structure is implicit Euler and yields

$$\mathbf{u}_{t+½\Delta t} = \mathbf{u}_{t-½\Delta t} + \Delta t\mathbf{a}_t\,, \tag{4.65}$$

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t\mathbf{u}_{t+½\Delta t}\,, \tag{4.66}$$

with the initial velocity offset given by an Euler step,

$$\mathbf{u}_{-½\Delta t} = \mathbf{u}_0 - \frac{1}{2}\Delta t\mathbf{a}_0\,. \tag{4.67}$$

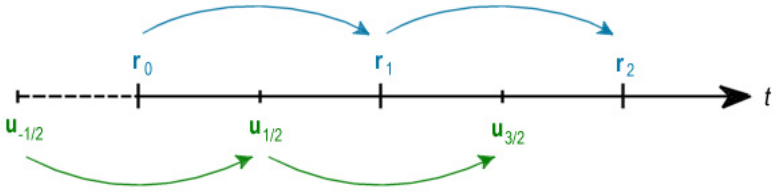


Figure 4.12 The leap-frog mechanism. The horizontal line represents time t, and the subscripts on the positions $\mathbf{r}$ and velocities $\mathbf{u}$ indicate the specific time.

The velocity at time t can be estimated by a simple midpoint approximation,

$$\mathbf{u}_t \approx \frac{\mathbf{u}_{t-½\Delta t} + \mathbf{u}_{t+½\Delta t}}{2}. \tag{4.68}$$

which is required when computing forces at time t.

4.5.4 Discussion

In theory, a time integration scheme will follow Newton's 1st law, but numerical dissipation can reluctantly damp the linear motion of the particles. Typically, this is not a problem in physics-based animation, because the damping can be explained as a small scale air resistance or friction. Especially the Verlet scheme is easily influenced by numerical damping. We have chosen not to introduce any explicit damping in the time integrators, due to the different ways integrators handle damping. We rely on the viscosity force to provide the necessary numerical damping.

4.6 Summery

In this chapter the following important topics are covered/achieved:

- Table 4.1 sums up the different densities, including all force densities and their expressions.
- The Navier-Stokes equations for an incompressible, isothermal Lagrangian fluid yields

$$\rho \frac{d\mathbf{u}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} .$$

- The acceleration of a particle, where $\mathbf{F}$ is the sum of internal and external force densities, becomes

$$\mathbf{a} = \frac{d\mathbf{u}}{dt} = \frac{\mathbf{F}}{\rho} .$$

- Particles are advanced through time by integrating the accelerations numerically using an integrator, such as the leap-frog method,

$$\mathbf{u}_{t+\frac{1}{2}\Delta t} = \mathbf{u}_{t-\frac{1}{2}\Delta t} + \Delta t \mathbf{a}_t ,$$

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{u}_{t+\frac{1}{2}\Delta t} ,$$

  where the velocity offset is given by

$$\mathbf{u}_{-\frac{1}{2}\Delta t} = \mathbf{u}_0 - \frac{1}{2}\Delta t \mathbf{a}_0 ,$$

  and the velocity at time t can be estimated by

$$\mathbf{u}_t \approx \frac{\mathbf{u}_{t-\frac{1}{2}\Delta t} + \mathbf{u}_{t+\frac{1}{2}\Delta t}}{2} .$$

- Implicit primitives are used as collision containers and obstacles, such as spheres and capsules, and allow for fast computations of contact points $\mathbf{cp}$, penetration depths d, and surface normals $\mathbf{n}$.
- A hybrid impulse-projection collision response method is applied to penetrating particles,

$$\mathbf{r}_i = \mathbf{cp} ,$$

$$\mathbf{u}_i = \mathbf{u}_i - \left(1 + c_R \frac{d}{\Delta t \|\mathbf{u}_i\|}\right)(\mathbf{u}_i \cdot \mathbf{n})\mathbf{n} .$$

Density	Expression (w/kernel)	Liquid/Gas
Mass	$\rho(\mathbf{r}_i) = \sum_j m_j W_{default}\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$	L + G
Surface normal (inward)	$\mathbf{n}(\mathbf{r}_i) = \sum_j \frac{m_j}{\rho_j} \nabla W_{default}\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$	L
Pressure force (internal)	$\mathbf{f}_i^{pressure} = -\rho_i \sum_{j \neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W_{pressure}\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$	L + G
Viscosity force (internal)	$\mathbf{f}_i^{viscosity} = \mu \sum_{j \neq i} \left(\mathbf{u}_j - \mathbf{u}_i\right) \frac{m_j}{\rho_j} \nabla^2 W_{viscosity}\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$	L + G
Gravity force (external)	$\mathbf{f}_i^{gravity} = \rho_i \mathbf{g}$	L
Buoyancy force (external)	$\mathbf{f}_i^{buoyancy} = b(\rho_i - \rho_0)\mathbf{g}$	G
Surface tension force (external)	$\mathbf{f}_i^{surface} = -\sigma \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|} \sum_j \frac{m_j}{\rho_j} \nabla^2 W_{default}\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$	L

Table 4.1 Densities and their expressions including smoothing kernel choice if employed. Last column informs whether the density applies for a liquid or a gas or both.

5 Implementation

The SPH method for Lagrangian fluid dynamics and all its auxiliary components described in Chapters 3 and 4 have been implemented in OpenTissue [28]. To retrieve a guaranteed working condition of the implementation that connects to this report, checkout revision **2310** from **TRUNK**, see Figure 5.1 for supplementary information about the revision. As of writing, OpenTissue employs Subversion as the version control system. For more details on how to get access to the repository please consult the OpenTissue web page.

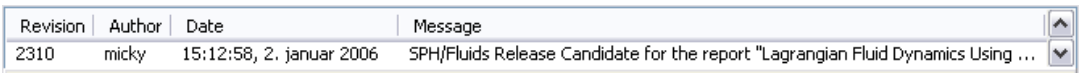


Figure 5.1 Screenshot of the log at rev. 2310 from the TRUNK.

We have included a running fluids application in OpenTissue, which demonstrates a complete utilization of SPH to simulate Lagrangian fluids. The application supports different fluid materials and different collision containers. Motion sequences grabbed from various fluid simulations, including the simulations presented throughout Chapter 6, are available on the OpenTissue media page.

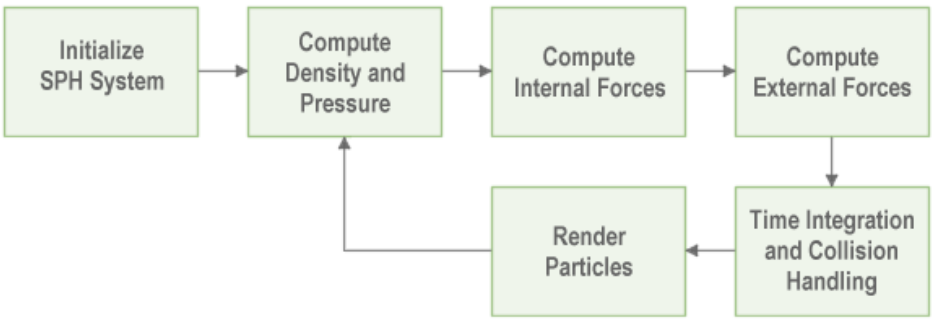


Figure 5.2 An overview of the Lagrangian fluid loop.

Figure 5.2 illustrates the basic simulation flow of our interactive Lagrangian fluid simulator. In this chapter we will go into details on the contents from each step. Section 5.6 describes the simulation flow in chronological order with references to employed equations and sections.

During the implementation of the SPH system and the fluid application we came across several performance issues. Most importantly is the time complexity for the SPH computations, which prevents any interactivity for fluid simulations of just a few hundred particles. This performance problem can be improved algorithmically, and the solution is described next. Also when working with physics-based animation, some parameter tweaking is unavoidable. Some time has also been put into a review of the important physical parameters that are a necessary evil to obtain realistic looking fluids.

We complete the chapter with a discussion on further improvements that can be applied to increase the overall performance of the method. The discussions are mostly non-algorithmically, but we also review one way to parallelize the SPH force computations.

5.1 Fast Nearest Neighbor Search

The evaluation of an SPH term iterates through all particles. As each particle must evaluate several SPH terms, the naïve time complexity for the Lagrangian fluid simulation with n particles is bound by $O\left(n^2\right)$. An asymptotic squared running time complexity is not satisfactory, and will in general deviate from any interactive speed as the amount the particles grow. The smoothing kernels all have the finite compact support radius h, and by definition the kernel contribution from any particle located beyond h is zero.

To increase the performance we must utilize that only particles near the location in question are relevant. Fortunately, the field of computational geometry provides us with a number of nearest neighbor search (NNS) algorithms, which are widely used to increase performance in collision detection [1]. A fast NNS algorithm typically subdivides space into a grid of voxels or cells, and a performance increase is obtained by limiting the search to the neighborhood in interest. The SPH computations can get a significant increase in performance by using a fast NNS algorithm to find the particles within a radius h. The asymptotic time complexity decreases from $O\left(n^2\right)$ to $O(nm)$, where m is the average number of particles found. Theoretically, if all particles are distributed uniformly, m is constant and thus the running time complexity will be linear in $O(n)$. However, for a linear time complexity in the amount of particles to hold, the search time from the NNS algorithm must be constant, such as the spatial hashing method.

5.1.1 Spatial Hashing

The spatial hashing method [36] is a fast NNS algorithm with a lookup that theoretically is bound by $O(1)$. It uses a hash function to generate hash keys for each grid cell. The hidden constant in this method is very much dependent of how well the hash function generates unique keys, and how fast hash keys are generated. Unique keys are important due to hash collisions, i.e. multiple keys that map to the same cell.

The hash function we are using can be found in [36] and maps a dicretized 3D point into a 1D hash index key, and it is defined by

$$hash(\hat{\mathbf{r}}) = \left(\hat{\mathbf{r}}_x p_1 \text{ xor } \hat{\mathbf{r}}_y p_2 \text{ xor } \hat{\mathbf{r}}_z p_3\right) \bmod n_H\,, \tag{5.1}$$

where n_H is the size of the hash table, and

$$\hat{\mathbf{r}}(\mathbf{r}) = \left(\lfloor \mathbf{r}_x / l \rfloor, \lfloor \mathbf{r}_y / l \rfloor, \lfloor \mathbf{r}_z / l \rfloor\right)^T \tag{5.2}$$

is the dicretized 3D point, with the cell size l. The three unknowns in (5.1) are large primes and in our case we are using

$$\begin{aligned} p_1 &= 73{,}856{,}093\ , \\ p_2 &= 19{,}349{,}663\ , \\ p_3 &= 83{,}492{,}791\ . \end{aligned} \tag{5.3}$$

Analyses and measurements of the hash table size, n_H, and the grid cell size, l, can be found in [36]. We cannot adopt the information explicitly for the fluid particles, because the analyses primarily concern collision detection between tetrahedra, but some findings are useful. The table size must be large enough to limit hash collisions, but at the same time, too large table sizes will increase memory cache misses. In all cases, the hash function (5.1) works most efficiently if the table size is a prime number [36].

Some information on vertices from [36] serves as a convincing foundation for our particles, and our own experiments verify the following findings. For the table size we use

$$n_H = prime(2n)\,, \tag{5.4}$$

where $prime(x),\ x \in \mathbb{Z}$ is a function that returns the next prime $\geq x$, and n is the amount of particles in our fluid simulation. We already know that $W(\mathbf{r}) = 0$ for $\|\mathbf{r}\| > h$ is valid for all smoothing kernels, thus for the cell size we use

$$l = h\,, \tag{5.5}$$

which must be the most optimal choice.

5.1.2 Spatial Particle Queries

The first pass in the spatial hashing is to insert all particles. Particle i is inserted into the spatial hashing table with the hash index returned by (5.1),

$$hash\_table\left[hash\left(\hat{\mathbf{r}}\left(\mathbf{r}_i\right)\right)\right] = Particle_i\ , \tag{5.6}$$

where each entry of $hash\_table$ must be a dynamic list that can hold multiple particles.

The second pass in the spatial hashing is to perform the particle queries. A single particle query first computes the discrete bounding box of the sphere represented by the location of the query particle, $\mathbf{r}_Q$, and the smoothing kernel radius h. The discrete bounding box of the sphere is represented by two vertices, a minimum and a maximum such as

$$BB_{min} = \hat{\mathbf{r}}\left(\mathbf{r}_Q - (h,h,h)^T\right) \quad , \quad BB_{max} = \hat{\mathbf{r}}\left(\mathbf{r}_Q + (h,h,h)^T\right). \tag{5.7}$$

Next we iterate from BB_{min} to BB_{max} over all three components, creating unique discrete positions. For each discrete position pos_D we retrieve a dynamic list L,

$$L = hash\_table\left[hash\left(pos_D\right)\right], \tag{5.8}$$

where L contains zero or more unique particles. Finally we check each particle j in L if it is inside the sphere using

$$\left\|\mathbf{r}_Q - \mathbf{r}_j\right\| \leq h\,, \tag{5.9}$$

and add it to a resulting particle container if (5.9) is true.

5.2 Incompressibility

The Navier-Stokes equations (2.1) and (2.2) describe the momentum of motion for the fluid, and require that the velocity field is divergence free for the fluid to be incompressible. However, the incompressibility equation assumes that the mass-density is constant in the fluid, but if it varies and becomes greater than the rest density ρ_0, we no longer have volume preservation, and hence the fluid is not incompressible.

For Lagrangian fluids the pressure force density is the only contribution that can secure the mass-density will not exceed the rest density anywhere. The incompressibility is thus implicitly modeled though the pressure term (4.12), which is repeated here for the sake of convenience

$$p = k(\rho - \rho_0). \tag{5.10}$$

The level of the fluid compressibility is controlled directly by adjusting the gas stiffness constant k, and incompressibility can only be obtained as $k \to \infty$. Unfortunately, as p in (5.10) works as a standard Hookean spring, increasing k implies decreasing the integration time step Δt, as numerical instabilities otherwise will occur. Another unfortunate side effect also appears when increasing

k. If $\rho < \rho_0$ the attraction force contribution from the pressure force will also increase, but this can be accounted for by using a smaller value for k in this situation.

We are not interested in using a too small integration time step, as it does not comply with the requirement of interactive fluid simulations. Consequently, using very large values for k in case of $\rho > \rho_0$ will not work as a solution for the lack of incompressibility. The moving-particle semi-implicit, or MPS, method from [17] is closely related to SPH and solves the incompressibility problem for Lagrangian fluids by applying additional mass-density and pressure correction terms. However, the correction terms must be solved simultaneously, and that usually requires an implicit procedure, e.g. the conjugate gradients method.

In [21] Monaghan states that near-incompressibility can be obtained by using an artificial state equations so that the level of compressibility is at or below 1%. The disadvantage is that the time step must then be a factor 10 shorter than normal. In our system near-incompressibility can thus be obtained by increasing k in (5.10) while using a time step of $\frac{1}{10}\Delta t$, which might just be acceptable.

5.2.1 Discussion

The compressibility of a fluid can be monitored using the mass-density. If the mass-density locally has become greater than the rest density, we know that the volume no longer is preserved for particles in that region. This is not the case in the opposite situation that is, when the mass-density is less than the rest density. It locally just indicates that we could be on the surface or in isolated droplets.

One possible explicit solution to the incompressibility problem is to look at particles more geometrically and employ a relaxation-based procedure. If a minimum distance between particles can be found that, when applied to all particles in a neighborhood, will guarantee that the mass-density locally never exceeds the rest density, it will be a matter of pushing particles away from each other until the rest density has been obtained globally. Such a solution could be applied to nearly incompressible fluid flows without decreasing the simulation time step.

Also, it might be beneficial to remember that SPH was not designed for incompressibility fluid flows, and instead include the continuity equation for a compressible fluid, under the assumption that the density no longer is constant. The continuity equation (4.6) then becomes the rate of change for the mass-density, and according to [21] the continuity equation for particle i can then be expressed in SPH formulation as

$$\frac{\partial \rho_i}{\partial t} = \sum_j m_j \big(\mathbf{u}_i - \mathbf{u}_j\big) \nabla W\big(\mathbf{r}_i - \mathbf{r}_j, h\big), \tag{5.11}$$

for some smoothing kernel W.

5.3 Physical Parameters

In this section we combine the theoretical and practical elements of the SPH method for Lagrangian fluid simulation. In the aid for using correct physical quantities, it is important not to exaggerate the realistic domain of the physical parameters. Some parameters are hard to define in reality, thus they are usually determined experimentally. However, we will try to describe their usage and what to focus on during the determinations. Interactive Lagrangian fluid dynamics can simulate different kinds of small-scale liquid materials, where rich details are in focus. The method is not meant to simulate large-scale fluids, e.g. open waters.

We will also put some focus on the physical units, which can have a great impact on a simulation if applied in different scales. The units used in this work are the standard System International (SI) units.

5.3.1 Fluid Volume and Particle Mass

Given a fluid, the volume, V in $[m^3]$, the fluid represents, and the fluid particle mass, m in $[kg]$, then the amount of particles that occupy the volume, n, can be determined using

$$n = \rho \frac{V}{m}, \tag{5.12}$$

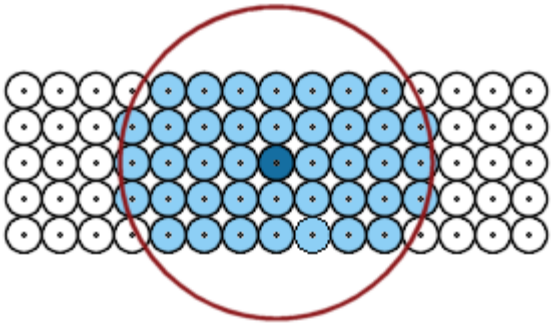
where ρ in $\left[\frac{kg}{m^3}\right]$ is the density of the fluid. The equation (5.12) is a direct rearrangement of (3.3), where it is defined for the entire fluid instead of a single particle. Often it is more practical to provide the total amount of fluid particles, and either the fluid volume or the particle mass, to determine the missing part. As an example, if $0.1\ m^3$ water should be approximated by $5{,}000$ particles, with the mundane density of non-boiling water at $1{,}000\ \frac{kg}{m^3}$, the mass for a single water particle yields

$$m = \rho \frac{V}{n} = 1{,}000 \frac{kg}{m^3} \frac{0.1 m^3}{5{,}000} = 0.02\, kg\,. \tag{5.13}$$

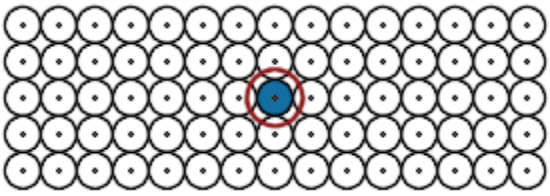
5.3.2 Smoothing Kernel Support Radius

The support radius, h in $[m]$, from a smoothing kernel is vital for a stable and robust fluid simulation. It is a normal misconstruction that as $h \to \infty$, then the SPH computations get more precise. In fact, if h is too large, the result from an SPH approximation can be very inaccurate. The simple explanation to why this is true is that the particles in real-time fluid simulations are sparsely. When h is large, the kernel weights particles near the center less than when h is small. Also, if h repeatedly reaches beyond the particles, the generated result will not be satisfactory. However, if $h \to 0$ the results will also be imprecise, due to not enough

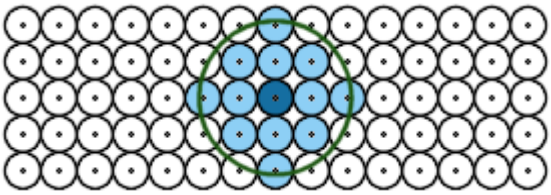
particles are included in the weighting by the smoothing kernel. A 2D analogue to the problem is depicted on Figure 5.3.



a) Too large support radius.



b) Too small support radius.



c) A usable support radius.

Figure 5.3 A 2D illustration of the problem of using either **a)** a too large support radius, or **b)** a too small support radius. The dark sphere in the center is the particle in question. The support radius is illustrated as a circle. In this example the support radius in **c)** will be a good choice.

More intuitively, the problem of finding the right support radius for a simulation can be converted into a problem of determining the number of particles an SPH calculation should include in average. If the support radius is considered as a sphere radius, we can calculate the size of a radius that would allow x particles to fill the volume of the sphere. We implicitly assume that the particle density is constant. This is not always true due to the pressure force fails in keeping the fluid incompressible, and thus the resulting radius size only becomes an average. Filling the volume of a sphere with x particles, by using the particle density of $\frac{n}{V}$, yields the sphere, and hence support radius,

$$\begin{aligned} x &= \left(\frac{n}{V}\right)\frac{4}{3}\pi h^3 \\ &\Updownarrow \\ h &= \sqrt[3]{\frac{3Vx}{4\pi n}}\ , \end{aligned} \tag{5.14}$$

where V is the fluid volume and n is the amount of particles occupying it.

Using (5.14) to determine a suitable support radius still leaves one unknown; the amount of the average kernel particles, x, which depends on several properties of the fluid, e.g. the strength of the total force acting on the fluid. In the time of writing we have yet to determine whether an expression that defines x can be constructed from the information that is already available. In our simulations we have chosen x

experimentally. Taking into account the problem of choosing a too large support radius, combined with the performance issues that raise when too many particles are included in each SPH calculation, then x should be chosen to be the smallest possible amount of particles that renders the fluid simulation stable, while still respecting the properties of the fluid material.

5.3.3 Time Integrator and Time Step

The time step, Δt in $[s]$, associated with the numerical time integration is very important for an interactive simulator. If the physical parameters are chosen to reflect reality best possible, and the integrator advances the fluid system in time using Δt, then Δt actually describes a discrete time step in the real world. The time step explicitly influences the experience of the fluid simulation. For the viewer to witness a fluid simulation in real time, the running frequency of the application must be $\frac{1}{\Delta t}\ Hz$, which equals $\frac{1}{\Delta t}$ frames per second (fps). Applications rarely simulate physics-based systems in real-time, which is why physics in e.g. computer games seems to run more or less in slow-motion.

During the development of the Lagrangian fluid simulator, we have been working with the three integrators discussed in Section 4.5. In all cases we have been able to use a time step of at least $\Delta t = 0.01\ s$. The Verlet integrator can in rare cases use a time step of $\Delta t = 0.02\ s$. However, collision handling is not working satisfactory as we manipulate positions rather than velocities, and this can be verified as small disturbances near the collision boundaries. The leap-frog integrator seems superior to the other integrators, regarding stability and performance. Employing the leap-frog integrator we can successfully simulate all types of fluids supported by our simulator, thus the leap-frog method will be our choice of time integrator.

We seek to use as large a time step as possible, and employing the Leap-frog integrator we can use a time step of $10\ ms$. The physical parameters that depend on the time step, i.e. the gas stiffness constant, must be adapted to respect the time step size. However, we do not see this as an additional drawback, compared to the gas stiffness constant itself. The fluid simulation can be paused at anytime by using $\Delta t = 0\ s$.

5.3.4 Gas Stiffness and Rest Density

The pressure force density is the most important contribution to a successful fluid simulation. Without the pressure force the fluid simply collapses. We already know that the pressure force cannot secure incompressibility, thus the task is to get as little compressibility as possible. The computation of the pressure force density includes the pressure at the particles, which also depend on the mass-densities.

The pressure term (5.10) employs a rest density, ρ_0 in $\left[\frac{kg}{m^3}\right]$, which can be chosen to reflect reality exactly. Besides using the rest density for the pressure calculations, we also use the same value as material density in the determinations of other fluid properties using (5.12). A physical correct fluid density depends on several other physical properties of the fluid, e.g. temperature, rest pressure, etc.,

and values for various fluid materials can be looked up in physics books and on the internet. As examples, water at $293.15° K$ ($20° C$) in atmospheric pressure at $101{,}325\ Pa$, has a density of $998.29\ \frac{kg}{m^3}$, and steam at $100{,}000\ Pa$ ($1\ Bar$) has a density of $0.59\ \frac{kg}{m^3}$.

As any other spring force (5.10) also employs a spring constant. The constant is the gas stiffness constant, k in $[J]$, also mentioned in Subsection 4.2.1. The gas stiffness constant is theoretically given by

$$k = nRT, \tag{5.15}$$

where n in $[mol]$ is the number of gas molecules, $R = 8.3144 \frac{J}{mol\,K}$ is the universal gas constant, and T in $[K]$ is the temperature. Unfortunately, we cannot use (5.15) to compute the gas stiffness. The result is simply too large to make any sense in a numerical simulation running at interactive rates. As an example for a 500 particle water simulation with a constant temperature at $293.15° K$, using the water particle mass of $0.02\ kg$, the gas stiffness yields

$$k = \frac{500 \cdot 0.02\,kg}{0.018 \frac{kg}{mol}} \cdot 8.3144 \frac{J}{mol\ K} \cdot 293.15\,K \approx 1{,}261{,}710\,J, \tag{5.16}$$

where the molar mass for water of $18.016\ \frac{g}{mol}$ has been looked up. A gas stiffness larger then 1×10^6 Joule will require an extremely small integration time step, which consequently will result in new numerical instabilities, not to mention a boring physics-based fluid animation.

We have chosen the gas stiffness to be a user defined material constant, which does not depend on the amount of particles used for a simulation. The gas stiffness is influenced by how viscous the fluid is, due to damping, and how large the time step is. In this work the time step is fixed globally, and the viscosity is fixed to reflect a specific fluid behavior. Tweaking the gas stiffness parameter to be as large as possible, while still keeping the fluid simulation stable under all conditions is a necessary evil. We have not successfully simulated any stable fluid material with a time step of $10\ ms$ and using a gas stiffness constant of more than $10\ J$.

5.3.5 Viscosity Coefficient

The viscosity force is essential in real-time fluid simulations. Physically, it simulates the viscous behavior of the fluid, but numerically, it damps the simulation and provides stability to the system. Basically, the more viscous the fluid is, the more stable the system becomes. In Section 4.5 different integration schemes are described. We have deliberately not added any damping terms to the time

integration, even though it is common to damp the velocity. The reason we have chosen not to use any explicit integration damping is because the magnitude of the damping coefficient very dependent of the chosen integration scheme. Instead we have chosen to unify the damping procedure by increasing the viscosity coefficient and thus use the viscosity force as the only damping term. The viscosity kernel has also been designed with damping in mind, and does allow omitting any kinds of additional damping [23].

The viscosity coefficient is the dynamic viscosity, $\mu > 0$ in $[Pa \cdot s]$, and to include a reasonable damping contribution, it should be chosen to be approximately a factor $1{,}000$ larger than any physical correct viscosity coefficient that can be looked up in the literature. However, care should be taken not to exaggerate the viscosity coefficient for fluid materials. If the contribution of the viscosity force density is too large, the net effect of the viscosity term will introduce energy into the system, rather than draining the system from energy as intended.

5.3.6 Surface Tension and Threshold

The surface tension coefficient can be chosen to reflect reality, but due to the asymmetrical inconvenience of the surface tension force, it is not easy to conclude whether or not the surface tension coefficient can be chosen completely arbitrary for artificial fluid materials. However, if the fluid is constrained in some container with only a single free surface, it is possible to detect the impact of the surface tension, and tweak it for to meet any desired criteria.

The surface tension force requires the computation of the inward unit surface normal. In (4.32) the threshold, ℓ, is used to identify whether or not a surface normal is fit for computation, and implicitly if the particle, used to compute the normal, is near or on the surface. We have not been able to connect any physically meaning to the threshold, thus is seems to be yet another user controlled constant. However, we have conducted experiments with the threshold value in several fluid simulations, and we have found a conceivable method to determine the threshold using already known properties of the fluid. We suggest the threshold can be obtained using

$$\ell = \sqrt{\frac{\rho}{x}}\,, \tag{5.17}$$

where ρ is the material density, or the rest density, of the fluid, and x is the amount of the average kernel particles, which is described in Subsection 5.3.2. If the threshold is determined using (5.17), its resulting SI unit does not make any physically sense, but our experiments reveal that it gives a plausible result in detecting the particles located on and close to the surface.

As a final note on the inward surface normal, once a normal has been detected as a surface normal using (4.32) and (5.17), the correct unit surface normal at particle i is given by

$$-\frac{\mathbf{n}(\mathbf{r}_i)}{\|\mathbf{n}(\mathbf{r}_i)\|}, \tag{5.18}$$

which might be of interest in the further developments, e.g. for visualization purposes.

5.4 Fluid Materials

Fluids are the common term for liquids and gasses. Examples of some of the interesting fluid materials we have been working with are summarized in this section. The important thing is the combination of the physical material parameters that together form the unique behavior of the fluid. The parameters are based on Section 5.3 together with experiments from our fluid simulator. As previously stated, we employ the pressure force from (4.10) and the leap-frog time integrator. Illustrations from the specific fluid types that are described in the following subsections are presented visually in Section 6.1.

Table 5.1 lists the parameters that are common for the different fluid materials. When a value is enclosed in parentheses, it is a value in an alternate unit. Values enclosed in parentheses in the material tables are physically correct, but deviates from the actual values used in the simulation of the fluid material. In Table 5.1 and in the forthcoming material tables a row with a description in italic means it is only of informational value, i.e. it is not used explicitly in the simulation.

Description	Symbol	Value	Unit
Gravitational acceleration	$\mathbf{g}$	$[0, 0, -9.82]^T$	$\frac{m}{s^2}$
Time step	Δt	0.01	s
Temperature	T	$293.15°$ $(20°)$	K (C)
Pressure	p	101325 (1)	Pa (Atm)

Table 5.1 Common properties used for all fluid materials.

As can be observed from the gravitational acceleration, we have chosen the negative z-axis as the direction "downwards". The temperature and pressure information has been used to obtain fluid properties, such as density, viscosity, and surface tension coefficient from the literature.

5.4.1 Water

Water is very liquid and brisk, but only a little viscid, thus it is a challenge keeping a realistic water simulation stable. We have modeled the water material by using as many physically correct values as possible. Table 5.2 lists the values used to comply with a realistic water simulation.

Description	Symbol	Value	Unit
Density (rest)	ρ_0	998.29	$\frac{kg}{m^3}$
Mass (particle)	m	0.02	kg
Buoyancy diffusion	b	0	n/a
Viscosity	μ	3.5 $\left(1.003 \times 10^{-3}\right)$	$Pa \cdot s$
Surface tension	σ	0.0728	$\frac{N}{m}$
Threshold	l	7.065	n/a
Gas stiffness	k	3	J
Restitution	c_R	0	n/a
Kernel particles	x	20	n/a
Support radius	h	0.0457	m

Table 5.2 Physical parameter values used in the realistic simulations of water.

5.4.2 Mucus

The mucus material is not modeled from a realistic fluid, but from our interpretation of such a substance. It is solely a material that resembles slime and goo as seen in motion pictures. Mucus is a highly viscous liquid with a strong surface tension. Table 5.3 lists the parameter values used to simulate our fictitious mucus.

Description	Symbol	Value	Unit
Density (rest)	ρ_0	1000	$\frac{kg}{m^3}$
Mass (particle)	m	0.04	kg
Buoyancy diffusion	b	0	n/a
Viscosity (dynamic)	μ	36	$Pa \cdot s$
Surface tension	σ	6	$\frac{N}{m}$
Threshold	l	5	n/a
Gas stiffness	k	5	J
Restitution	c_R	0.5	n/a
Kernel particles	x	40	n/a
Support radius	h	0.0726	m

Table 5.3 Physical parameter values used in the simulations of our own mucus material.

5.4.3 Steam

Realistic hot water steam undergoes diffusion and dissipation, which we do not model explicitly. Buoyancy is also common for a gas, due to the diffusion of the temperature. We only work with isothermal fluids, thus the buoyant forces we use are based upon the difference in mass-densities, and cannot model the turbulence often associated with a hot gas. Table 5.4 lists the values used to imitate a steam.

Description	Symbol	Value	Unit
Density (rest)	ρ_0	0.59	$\frac{kg}{m^3}$
Mass (particle)	m	5×10^{-5}	kg
Buoyancy diffusion	b	5	n/a
Viscosity (dynamic)	μ	0.01 $\left(1.2 \times 10^{-5}\right)$	$Pa \cdot s$
Surface tension	σ	0	$\frac{N}{m}$
Threshold	l	-	n/a
Gas stiffness	k	4	J
Restitution	c_R	0	n/a
Kernel particles	x	12	n/a
Support radius	h	0.0624	m

Table 5.4 Physical parameter values used in the simulations of isothermal steam.

The normal gravity force is disabled when the buoyancy force is applied. This is a practicable hack when simulating gasses. Either gravity is turned off or substituted by some external buoyant force.

5.5 Rendering

We have not included any surface rendering techniques in this report, as we believe that the standard visualization methods have been reviewed thoroughly in the literature. However, we have implemented a fluid surface visualization prototype that is based on the marching cubes algorithm [19]. A particle-based gas could probably be visualized using translucent sprites or blobs, thus no surface extraction is necessary. Section 7.2 contains more information on our work in progress for alternate fluid surface visualization methods.

To visualize the fluid materials we render the particles as colored spheres. This gives a believable understanding of how the fluid flows. Where the mass-density is high the concentration of fluid particles is larger compared to areas with a low mass-density. Using a fixed sphere radius for all particles will in general not look reasonable, as either gaps or overlaps between spheres will occur. The mass-densities of the particles can be used to determine the sphere radii convincingly. Using the relation given by (3.3) the sphere radius r can be computed as

$$\begin{aligned} V &= \frac{4}{3}\pi r^3 \\ \frac{m}{\rho} &= \frac{4}{3}\pi r^3 \\ r &= \sqrt[3]{\frac{3m}{4\pi\rho}}. \end{aligned} \tag{5.19}$$

To linearly scale the mapping between the sphere radius and the mass-density at the particle, we multiply the right hand side of (5.19) by a user defined scalar constant $s > 0$,

$$r = s\left(\sqrt[3]{\frac{3m}{4\pi\rho}}\right). \tag{5.20}$$

5.6 The Lagrangian Fluid Method

In this section we will summarize the different parts of the simulator with references to the presented theory and the implementation details. Figure 5.2 gives a graphical overview of the entire simulation flow. Each "box" will be described in details in the following subsections. Although the figure can be interpreted as the internal forces must be computed before the external forces, this is not strictly required. The internal and external force density computations can be performed in any order.

The subsections include algorithmically step-by-step procedures, which can be used as an implementation layout, followed by additional comments on the steps.

5.6.1 Initialize SPH System

i. Create the fluid material, e.g. use as reference the values given in Section 5.4.
ii. Create n particles and set the positions, the initial velocities, and the fixed particle mass, e.g. use (5.12) to determine missing values.
iii. Initialize the smoothing kernels using (5.14) to compute the compact support radius.
iv. Create the spatial hashing data structure using (5.4) and (5.5), and insert each particle using (5.6).
v. Create collision objects, e.g. use the implicit primitives provided in Subsection 4.4.2.
vi. Initialize the leap-frog integrator using (4.67) for all particles.

In step vi the leap-frog initialization requires the particle acceleration at time $t = 0$ to compute the velocity offset at time $t = -½\Delta t$. The initial acceleration can be computed by (4.2), where the forces working on the particles are computed in the next three subsections.

The particles must all have unique positions, i.e. no two particles must share the same location. Particles having the exact same positions will never be separated. They will wrongfully be treated as one larger particle and thus result in oddly behaved fluid dynamics. It is also important to know that if the particles all share the same value for the same component, then the dynamics is constraint to work in two dimensions. The same is valid if two components are the same for all particles, then the simulator only works in one dimension. Only external influences can break

this constraint, e.g. a collision handling that due to the collision response can change the particle positions, and hence the equal component values.

5.6.2 Compute Density and Pressure

For each particle i, do:

i. Search for the particle neighborhood N_i using the spatial hashing single particle query from Subsection 5.1.2.

ii. Compute mass-density ρ_i using (4.6), but only iterate over the particles from N_i.

iii. Compute pressure p_i using (4.12), and use the material density as the rest density, ρ_0.

5.6.3 Compute Internal Forces

For each particle i, do:

i. Search for the particle neighborhood N_i using the spatial hashing single particle query from Subsection 5.1.2, and use only the particles from N_i to compute the SPH forces.

ii. Compute the pressure force density acting on the particle using (4.10).

iii. Compute the viscosity force density acting on the particle using (4.17).

iv. $f_i^{internal} \leftarrow f_i^{pressure} + f_i^{viscosity}$.

In step i we need to search for the particle neighborhood once more, because all SPH forces depend on the mass-density that must be computed before any force density.

5.6.4 Compute External Forces

For each particle i of a liquid fluid, do:

i. Compute the gravity force density using (4.24).

ii. Compute the inward surface normal using (4.28).

iii. $f_i^{surface} \leftarrow \mathbf{0}$.

iv. If (4.32) is true using (5.17) as the threshold then

   a. Compute the surface tension force for the particle using (4.26).

v. $f_i^{external} \leftarrow f_i^{gravity} + f_i^{surface}$.

For each particle i of a gaseous fluid, do:

i. Compute the buoyancy force density using (4.25).

ii. $f_i^{external} \leftarrow f_i^{buoyancy}$.

The steps ii and iv.a for the liquid fluid also depend on the particle neighborhood N_i, as both steps employ SPH. Fortunately, there are no other dependencies that

will disallow the steps to continue from step iv in the previous subsection, and thus to reuse the particle neighborhood N_i that has already been computed.

5.6.5 Time Integration and Collision Handling

For each particle i, do:

i. $F_i \leftarrow f_i^{internal} + f_i^{external}$.

ii. Compute the particle acceleration $\mathbf{a}_i$ using (4.2).

iii. Use the leap-frog integrator to advance particle velocity and position using (4.65) and (4.66), respectively.

iv. Perform collision detection against collision primitives using (4.33).

v. If a collision occurred then

   a. Project particle position according to the contact point using (4.55).

   b. Update the velocity using (4.58).

vi. Approximate the new particle velocity using (4.68).

During the collision detection in step iv all implicit primitives are tested for penetrations. As soon as a collision is detected the collision is handled in step v. If multiple collision objects are included in the simulation, unhandled collisions might occur, e.g. with overlapping implicit primitives or implicit primitives contained in each other. One solution to this problem includes only to update the position in step v and then return to step iv and check for further collisions. For each collision the position is only updated in step v.a if the collision has a larger penetration depth than already handled. By default, the larger the penetration depth is, the closer to the previous position the collision occurred. When no further collisions are detected, the velocity update from step v.b can be performed using the collision data associated with the largest penetration depth.

Once all the particles have been updated the spatial hashing must likewise be updated to reflect the new particle positions.

5.6.6 Render Particles

For each particle i, do:

i. Render a sphere with its center at $\mathbf{r}_i$ and a radius determined by (5.20).

The color of the spheres could be stored with the fluid material, e.g. some blue shade for water, and a green slimy shade for the mucus material.

5.7 Discussion

These discussions are regarding issues and improvements due to performance. As of writing, the SPH implementation and demonstration application have only been numerically optimized on a small scale, as we have been focusing at optimizations

on an algorithmic level. There are several things that can be done to help gain a higher performance frequency, some of which will be discussed in this section.

In Section 5.6 we perform the same particle query twice for each particle, due to particle dependencies between mass-density and SPH forces. Although the asymptotic running time for a single particle query is bound by $O(m)$, m being the average number of particles returned by the spatial hashing, the hidden constant in the O–notation has some impact on the performance of the application. It will increase performance if only a single particle query is necessary per particle. A multiple particles query employs the single particle query procedure from Subsection 5.1.2 for each query particle. The problem is that we have no longer any idea about the connectivity between the query particles and resulting particles. Instead when (5.9) is valid we can add the particle pair $\langle Particle_Q, Particle_j \rangle$ to the resulting particle container, where $Particle_Q$ is the current query particle. This restores the lacking particle connectivity, but it also increases the particle query memory consumption from $O(m)$ to $O(nm)$.

The fluid solver presented in Section 5.6 handles each particle sequentially, i.e. all forces acting on a particle are computed before the next particle iteration is commenced. The reason for the sequential behavior is the single particle query, but employing the multiple particles query we can now allow for a parallel solver. To handle all the particles in parallel does in general imply that we can complete the computations of the same force for all particles before the next one, i.e. we can iterate over the forces. This is an important achievement that can allow the computations to be performed on a distributed system. For our specific purpose it will allow the fluid solver to be implemented on a multiprocessor architecture or on a graphics processing unit (GPU).

Memory management and cache coherency are important terms when working with many particles and only expect a linear performance decrease with respect to a linear increase in particles. The spatial hashing data structure and the resulting particle query container store references to all the particles, which are not optimal for the cache coherency. Due to cache misses we expect a better performance if particle indices are used rather than particle references. The indices can be used to retrieve particles from the same place in memory. In [23] a simulation speed-up of a factor 10 was obtained by storing copies of the particles in their grid data structure, and thus doubling the memory consumption. We store a number of attributes with each particle, and experiments have shown that using copies of the particles in the spatial hashing grid cause a noticeable decrease in performance, due to the duplication of all the particle members.

Assuming that particle indices are employed for the spatial hashing and the result container instead of particle references, and that the multiple particles query is used for neighborhood retrieval, then another performance gain can be obtained without much trouble. Forces between pairs of particles have in general the same magnitude, but with opposite directions. This implies that the force contribution between particle pairs only needs to be computed once. Using the multiple particles query each index pair is represented twice, i.e. $\langle i, j \rangle$ and $\langle j, i \rangle$. Taking advantages of

Newton's 3rd law, a straight forward method to omit the duplicates is to add the index pair to the result container only if $j < i$. Up next, the SPH forces between each particle pair is computed and added to both particles, but with different signs.

Many mathematical operations are executed each frame to perform comparisons, calculate constants, etc., and several of these can be precomputed in the initial setup of the simulation, e.g. the constants used in the computations of the smoothing kernels, as the support radius is kept constant throughout the simulation. Even kernels can be precomputed and stored in arrays. The precision of a precomputed kernel is directly dependent of the size of the array. The left and right hand sides in comparisons can be squared if both sides are positive, because $x \leq y \leftrightarrow x^2 \leq y^2$ if $x, y \geq 0$. This little trick can ease the computations in many comparisons that often require the evaluation of the square root. A classic example is the particle-inside-sphere comparison from (5.9), which is used to decide if a particle is within the range of the compact support radius. The comparison becomes cheaper if both sides are squared,

$$\left(\mathbf{r}_Q - \mathbf{r}_j\right) \cdot \left(\mathbf{r}_Q - \mathbf{r}_j\right) \leq h^2 . \tag{5.21}$$

5.8 Summery

In this chapter the following important topics are covered/achieved:

- The SPH computational time complexity of $O\left(n^2\right)$ for n particles can be reduced to $O(mn)$, m being the average number of interacting particles, by utilizing a spatial hashing method for fast retrieval of the nearest particle neighborhood.
- Near-incompressibility can be obtained by decreasing the time step to $\frac{1}{10}\Delta t$ and increasing the gas stiffness constant as much as possible, while still obtaining a stable simulation.
- The relation between fluid volume, fluid particle mass, and the amount of fluid particles is
$$n = \rho \frac{V}{m} .$$
- The compact kernel support radius can be determined intuitively by choosing x, which is the average amount of interacting particles,
$$h = \sqrt[3]{\frac{3Vx}{4\pi n}} .$$
- The leap-frog time integrator is superior to the Verlet and semi-implicit Euler integrators, and allows for a simulation time step of $\Delta t = 0.01s$.
- The viscosity coefficient is also used for numerical damping in the system, and should be chosen to be approximately $1{,}000$ times larger than true values.

- As physically correct fluid properties as possible are used to define our three fluid materials, which include water, mucus, and steam.
- Fluid particles are rendered as colored spheres with a radius determined by the particle mass-density,

$$r = s\left(\sqrt[3]{\frac{3m}{4\pi\rho}}\right).$$

- A complete algorithmically procedure to implement our Lagrangian fluid solver step-by-step is presented using a high level pseudo language.
- The default sequentially fluid solver can be converted into a parallel solver by using the multiple particles query rather than the single particle query.

6 Results

In this chapter we will put our Lagrangian fluid solver to the test, and present some of the more interesting results that we have obtained. We will test different aspects of the topics that we have described through Chapter 4 and 5. The majority of the test results and conclusions will be based on visual convictions. We are computer scientists, thus we are in general pleased if the results are visually convincing, while keeping in mind that the simulations already use as physically correct fluid parameters as possible. In this chapter we depict still frames from motion sequences grabbed from fluid simulations. The motion sequences can be found on the OpenTissue media page [28].

Firstly, we present various situations of our fluids and their properties, along with some cases of user interactivities. Next, we will go into more details on the fluid flows, and try to frame a survey on whether the fluid flows are advanced realistically. Subsequently, stability and performance tests are executed which are important matters regarding interactive fluid design. Finally, we reveal the darker side of the method, which primarily concerns the lack of incompressibility.

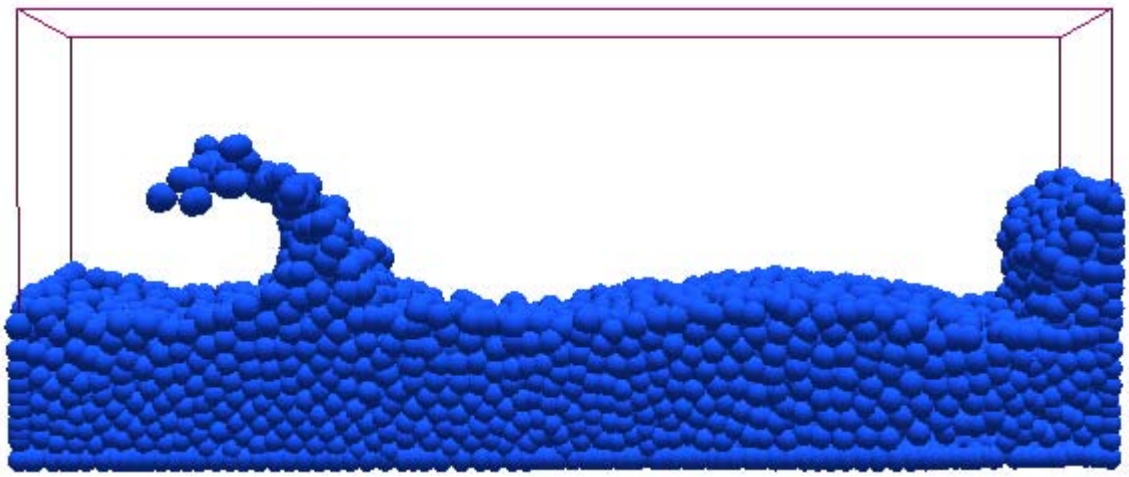
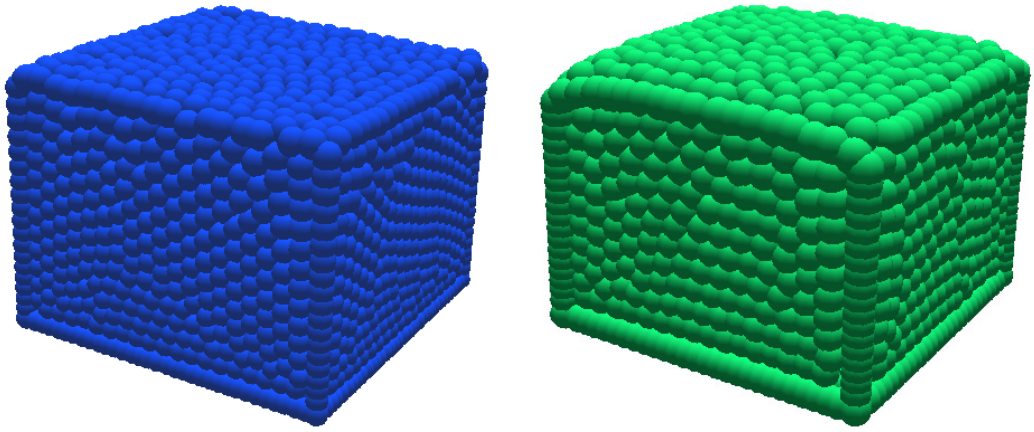


Figure 6.1 Lagrangian water constrained in a rectangular box. The complex wave to the left is a result of two currents that met in high speed.

6.1 Fluid Properties

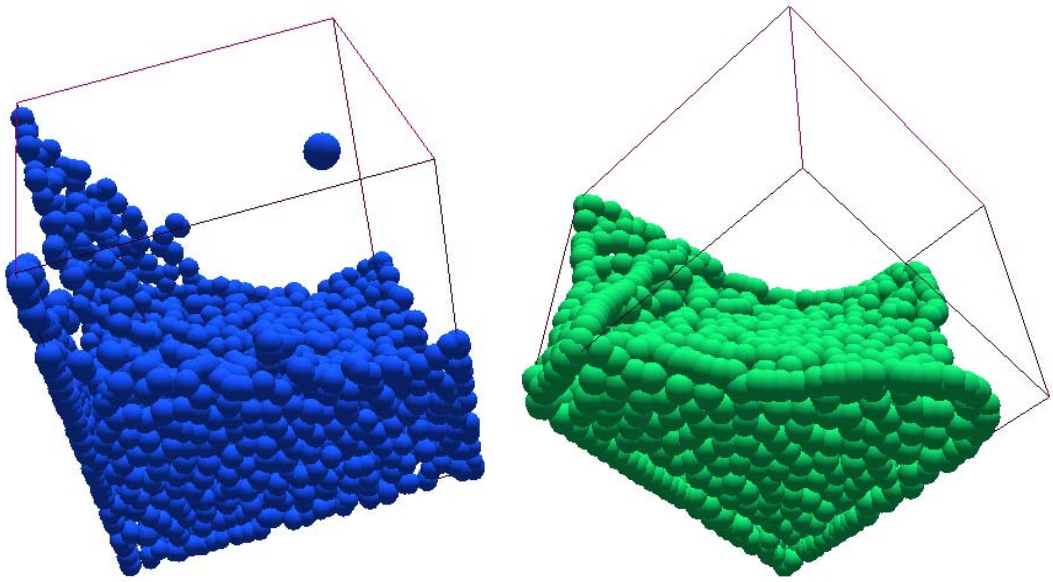
The fluid properties of water and mucus, presented in Subsection 5.4.1 and 5.4.2, respectively, are used throughout the simulations illustrated in this section. Figure 6.2 depicts both liquids for approximately the same volume. The water particle mass is half the mass of a mucus particle, which results in twice the amount of particles for the water, compared to the mucus. The interesting part on Figure 6.2 is the difference in surface tension, which cause the different fluid surface curvatures. On Figure 6.3 another difference between the fluids is depicted. A low viscosity

causes water to react more briskly in the rotating mixer as opposed to mucus, which has a high viscosity.



a) Water, low surface tension. **b)** Mucus, high surface tension.

Figure 6.2 Fluids with different surface tension coefficients.



a) Water mixer, low water viscosity. **b)** Mucus mixer, high mucus viscosity.

Figure 6.3 Fluids with different viscosity are tumbled around in rotating mixers.

In Subsection 5.4.3 we have described the parameters for a steam material. The simulation of a steam is depicted on Figure 6.4. The scene consists of $2,000$ particles running at approximately 20 frames per second. We are aware that the steam motion is not modeled realistically, because the thermal buoyancy effect has been omitted from the model. Instead, we have based an artificial buoyancy force on the difference of mass-densities, which can be considered a hack. The steam material is merely present to illustrate that it is possible to use our fluid solver to simulate other fluids than liquids, although it requires an additional development of a thermal field using SPH.

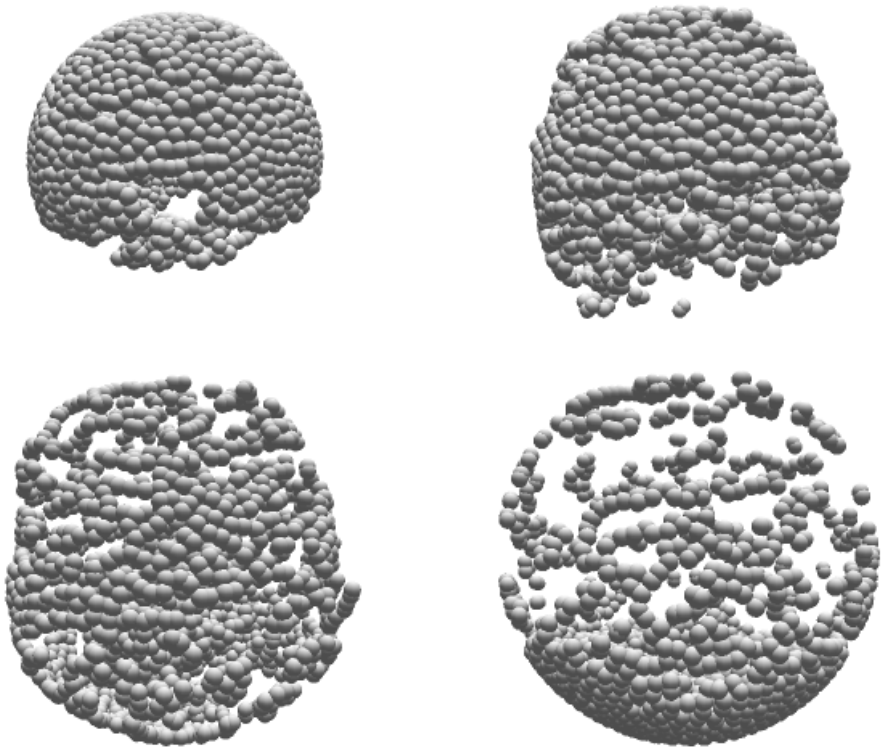


Figure 6.4 A simulation of steam (or smoke) that expands inside a sphere container. The sequence is over time from left to right, top to bottom.

User interactions with the fluid are an important part of interactivity. We allow a user to interact with a fluid by performing transformations to the fluid container. On Figure 6.5 a user shakes the fluid capsule, which has been rendered invisible to focus on the water. Nice water characteristics, e.g. the splashing, appear, due to the external user influence. The invisible mucus capsule on Figure 6.6 is also shaken, but unlike water the mucus material does not splash. Both simulations run interactively with $1,500$ particles.

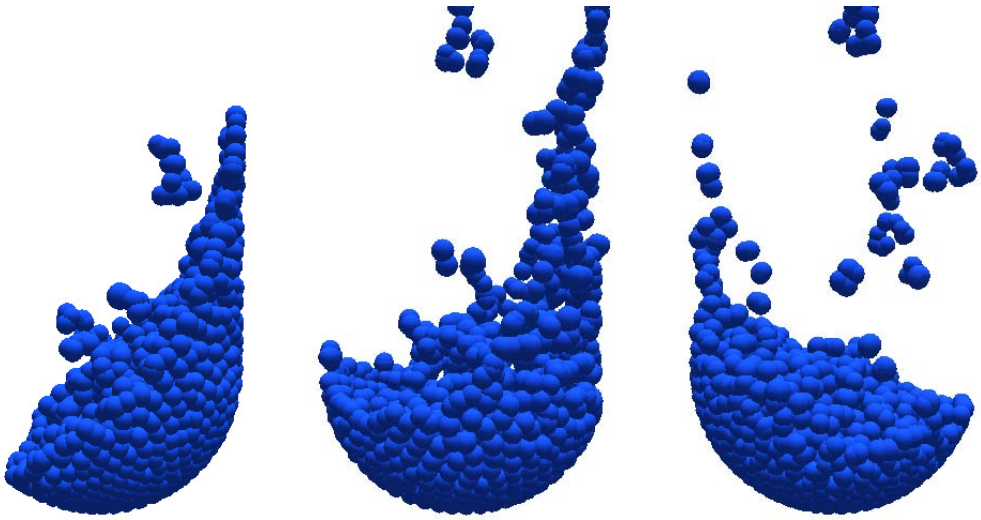


Figure 6.5 The fluid container is pulled quickly from right to left causing the water to splash.

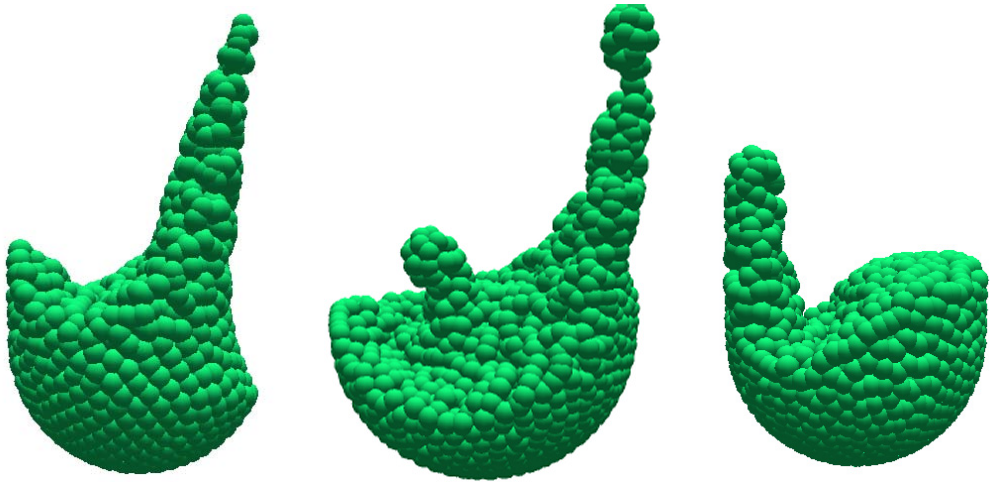
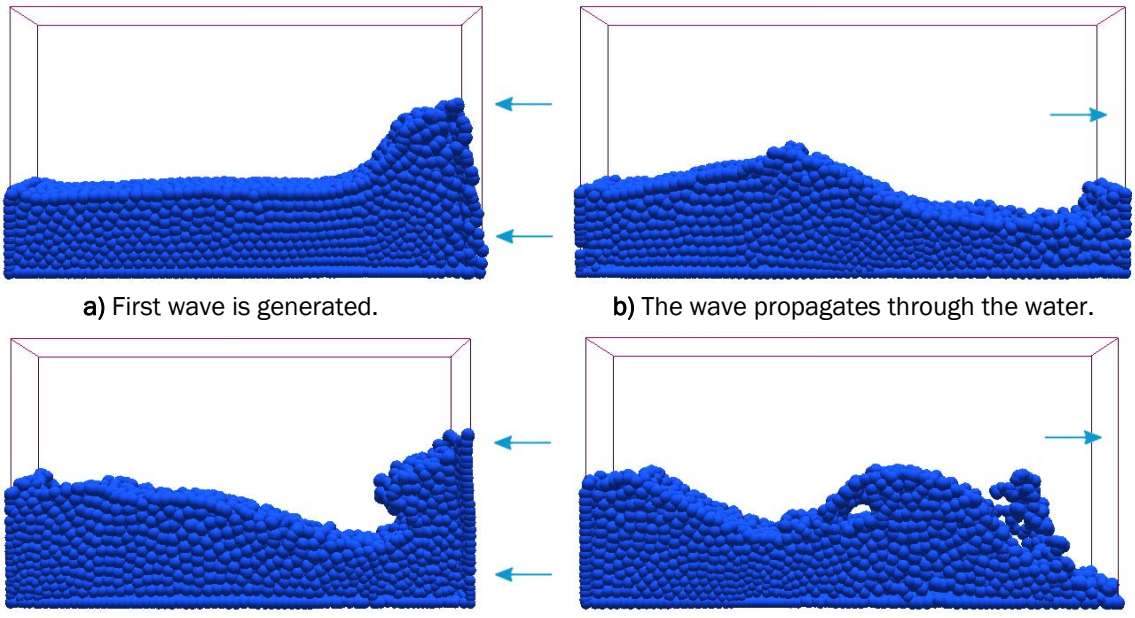


Figure 6.6 The fluid container is pulled quickly from side to side causing the mucus to create long connected blobby tails. Due to the high viscosity and surface tension the mucus fluid cannot splash like water.

Waves can be generated in several ways, e.g. by providing an external wind force to the fluid surface, but we have created a special wave container for the same purpose. The container is a rectangular box that can generate waves by moving one of its faces. The moving box face is controlled by a positive sine function over time, i.e. $|\sin(t)|$, where t is the time. This will generate a realistic face motion to produce waves. Figure 6.7 illustrates 4 still frames from a motion sequence that simulates $4,400$ water particles in the wave box. The frame on Figure 6.1 is also from a similar water wave simulation. The fluid solver runs the scene at approximate 5 frames per second, including the visualization of the particles.



a) First wave is generated.

b) The wave propagates through the water.

c) Second wave is launched.

d) The wave breaks when it collides with the first.

Figure 6.7 Water waves generated in a rectangular water tank similar to a wave pool in a water park. The arrows indicate the box face movement that generates the waves.

6.2 Fluid Flows

The analysis of fluid flow is more an area of interest for physicists than computer scientists. However, in order to be convinced that the Lagrangian fluid method can produce realistic fluid motion we will examine the fluid flow.

We will study the velocity flows produced by the dam-break problem for the water and mucus materials. In a classic dam-break problem the fluid is constrained inside a dam, and when the fluid is at rest the dam is broken, or the barricade that constrains the fluid is removed. The fluid now flows freely and often collides with a vertical wall.

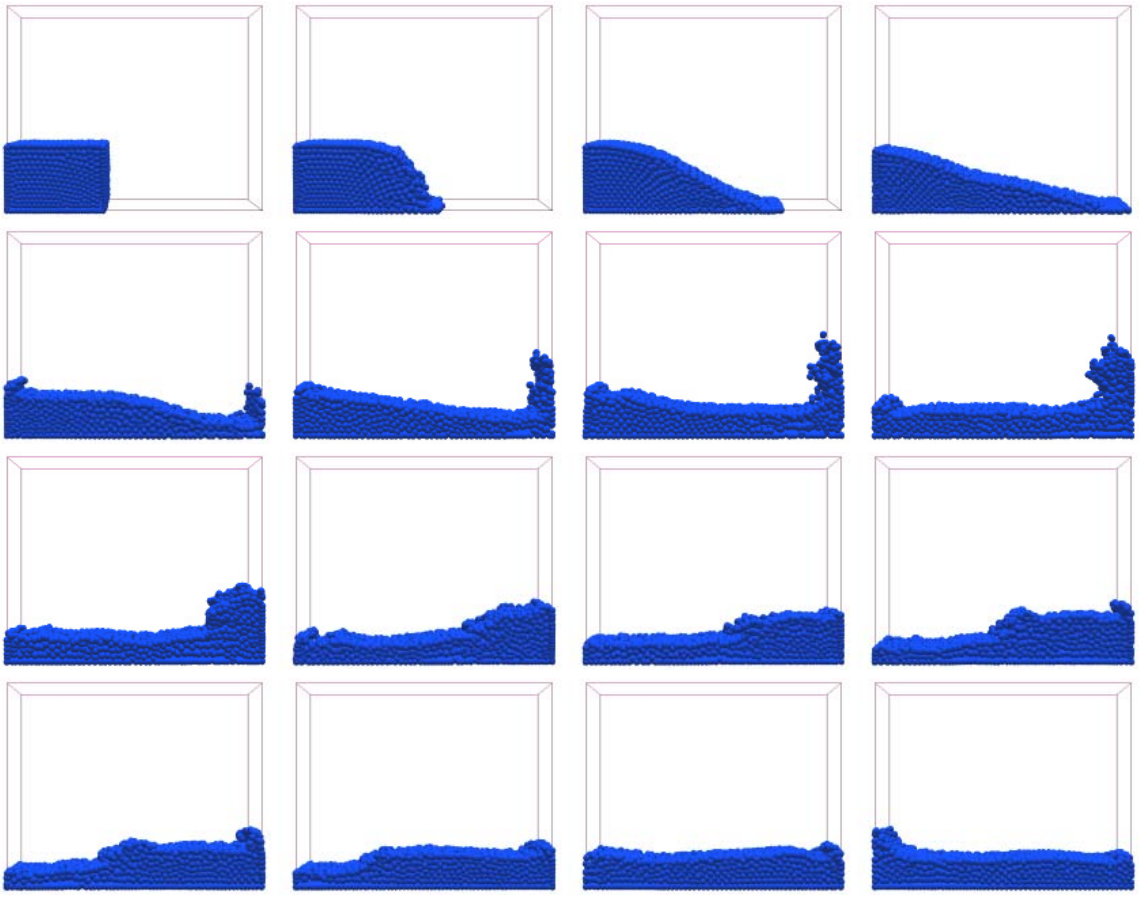
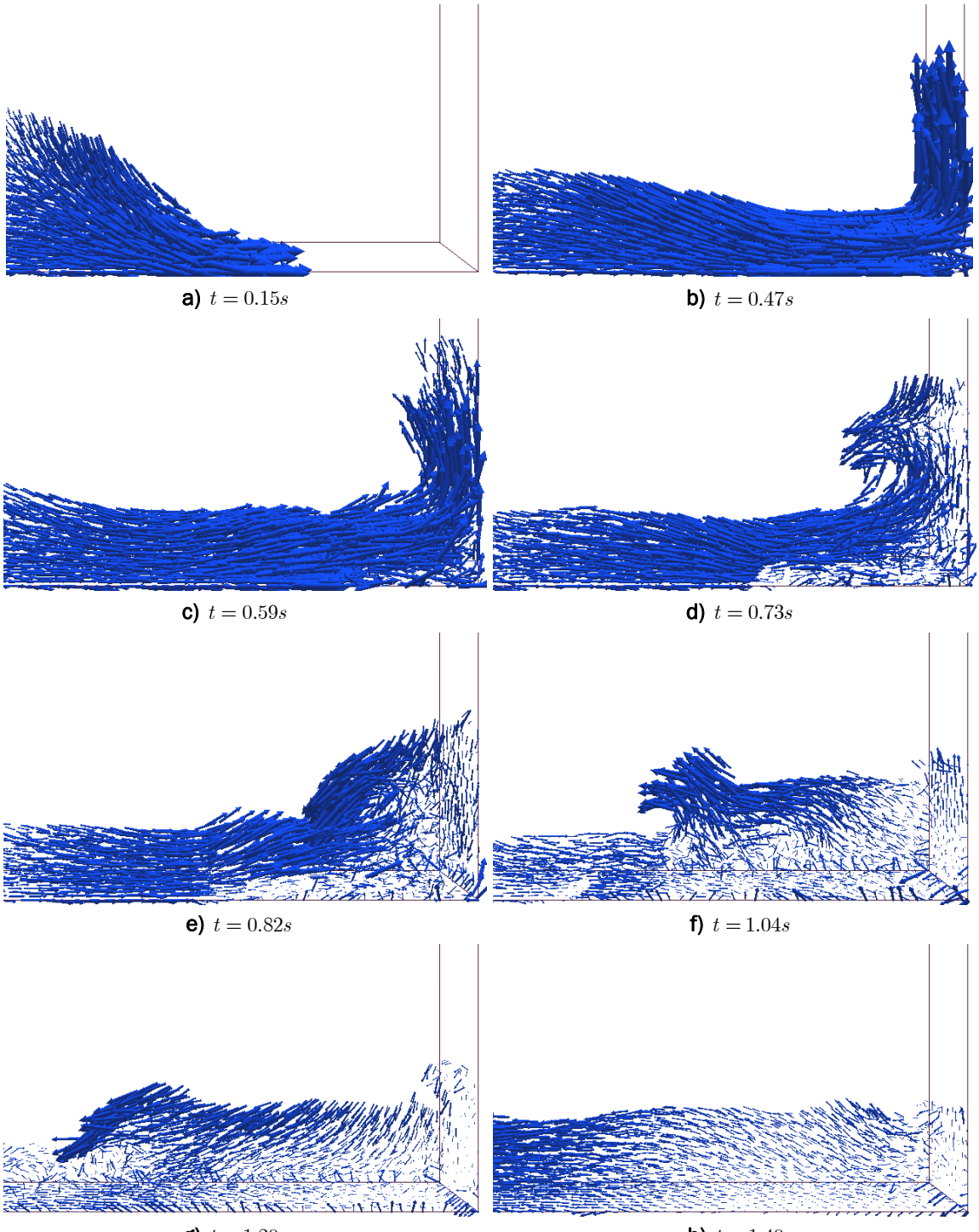


Figure 6.8 Dam-break flow of water. The simulation time interval is $0.1s$ between each frame, from left to right, top to bottom.

Frames from the dam-break of water simulated by $2{,}250$ particles are depicted on Figure 6.8. This is just a survey of how the visible water particles flow in the dam-break problem. On Figure 6.9 we have focused on the water flow. The flow lines are particle velocities visualized as arrows. The arrows are scaled according to the magnitudes of their velocity vectors. In **a)** the dam is broken, and the water is sliding down and increasing the velocity at the smooth bottom. In **b)** the water has collided with the wall, and the particles are pushed upwards due to the pressure from the water at high speed. In **c)** the water begins to overturn and in **d)** a wave is forming with the help from the small amount of water above it that is falling down. The wave

hits the underlying water in **e)** and creates a small splash in **f)** due to the impact with the oncoming current. The splash starts another wave in **g)** and finally in **h)** the current is moving away. From **c)** notice how the water flow is returning at the bottom after the collision, which manifests itself in thinning out the velocity strengths on its way.



a) $t = 0.15s$ **b)** $t = 0.47s$ **c)** $t = 0.59s$ **d)** $t = 0.73s$ **e)** $t = 0.82s$ **f)** $t = 1.04s$ **g)** $t = 1.20s$ **h)** $t = 1.40s$

Figure 6.9 Close-up on the water velocity flow from the dam-break simulation on Figure 6.8. Precise simulation times are indicated below each frame.

We have compared the flow lines on Figure 6.9 with the dam-break problems discussed in [3]. Although the paper focuses on interfacial flows, the dam-break examples are a great reference for comparisons. Without going into physical details we will state that the flow lines generated by our fluid solver seem very convincing in describing how water flows.

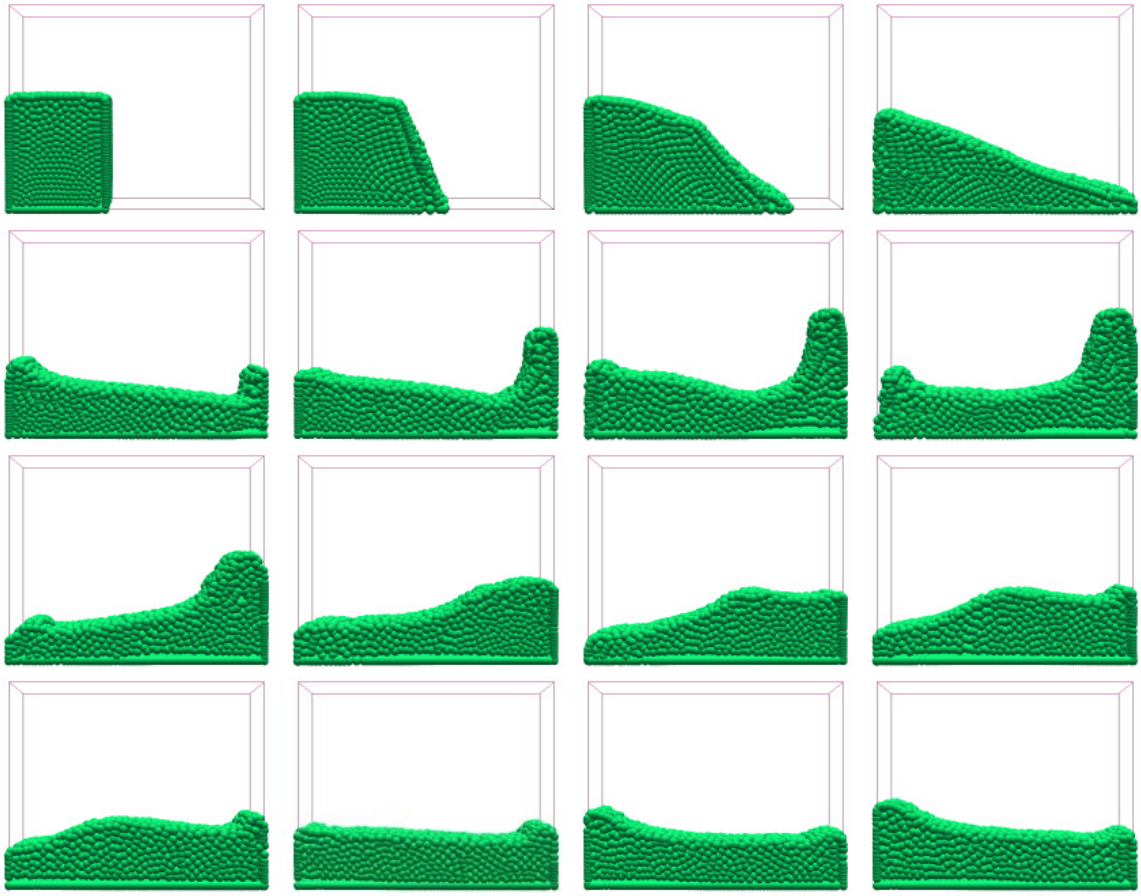


Figure 6.10 Dam-break flow of mucus. The simulation time interval is $0.1s$ between each frame, from left to right, top to bottom.

A mucus dam-break simulation of $2,250$ particles is depicted on Figure 6.10. The mucus material is very viscous, and it is easy noticeable that the mucus flow is much calmer compared to the water flow. On Figure 6.11 the focus is on the mucus velocity flow from the simulation depicted on Figure 6.10. The flow lines are common for a viscous fluid flow. In **a)** the dam is broken, and as in **b)** the front of the mucus flows more freely, compared to the rest of the mucus fluid that due to the high viscosity partly still possesses the initial mucus dam structure. The mucus flow has collided with the vertical wall in **c)** and the flow is propagating upwards. The turning point for the mucus fluid happens around **d)** where it starts to ebb again without creating an overturn wave as in the water dam-break simulation. In **e)** the mucus hits the oncoming fluid current and a broad wave is forming. In **f)**, **g)**, and **h)** the broad wave is slowing transporting away.

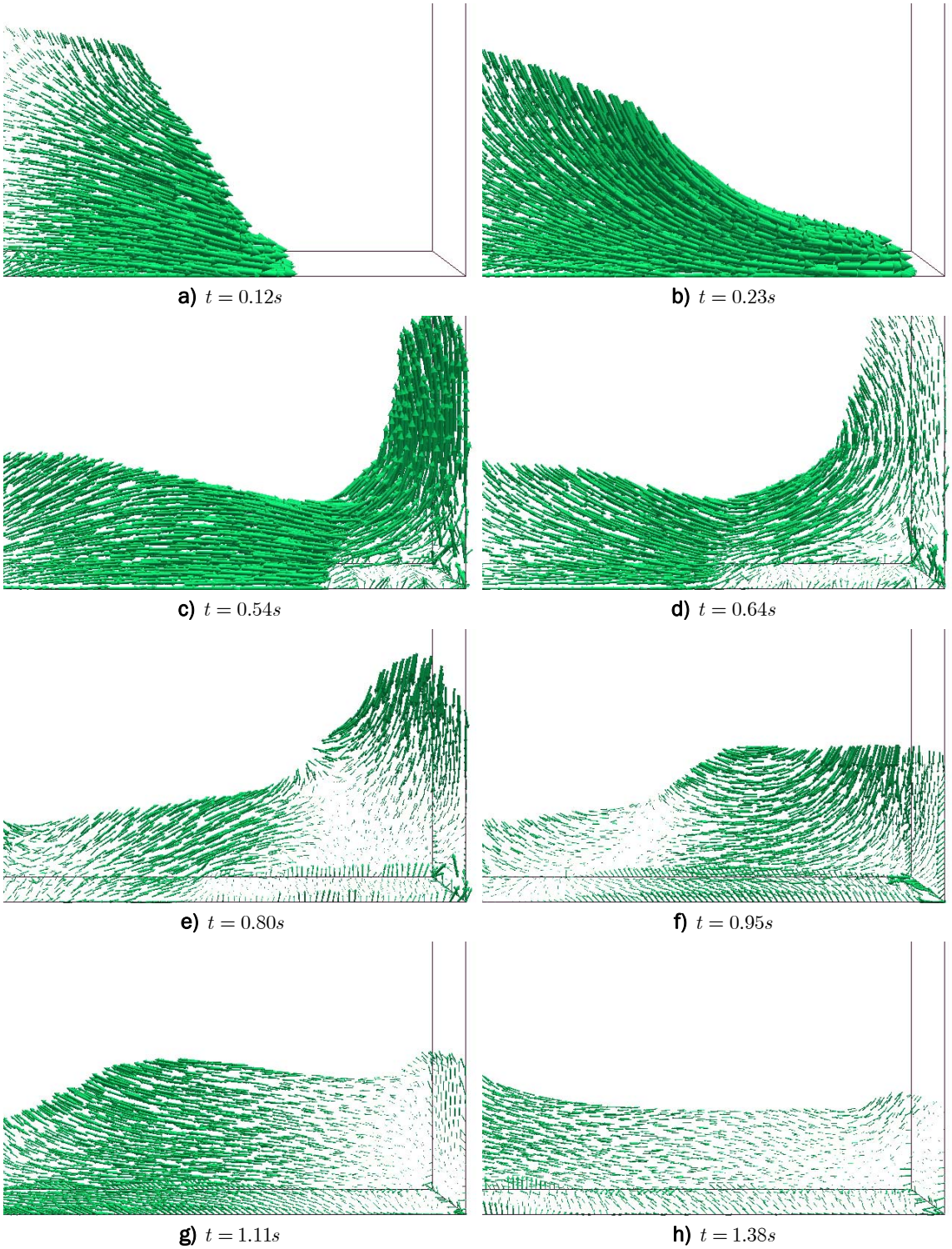
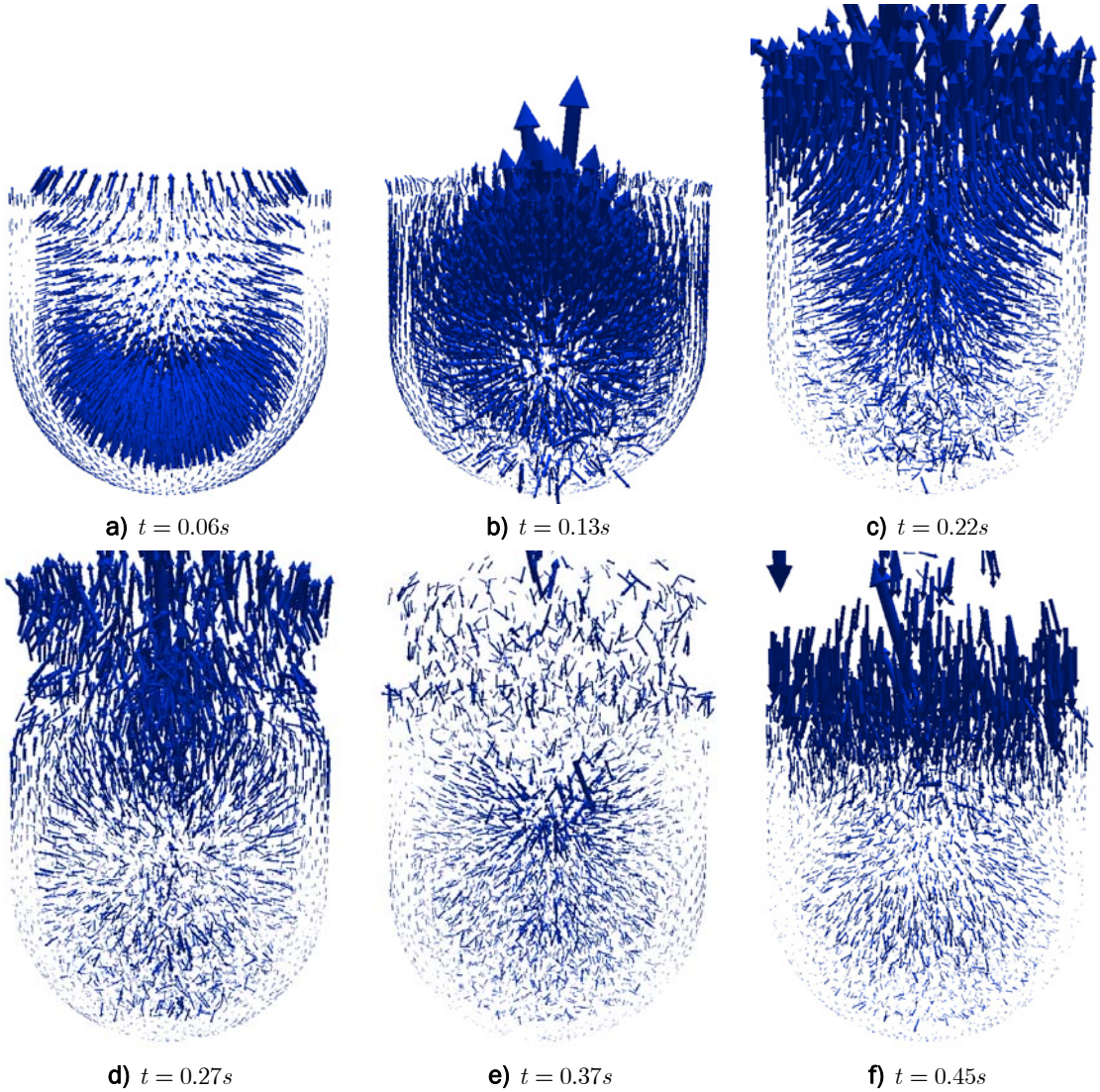


Figure 6.11 Close-up on the mucus velocity flow from the dam-break simulation on Figure 6.10. Precise simulation times are indicated below each frame.

Shock waves have been studied by physicists for different kinds of fluids. We have simulated a vertical water pressure shock and will look at the flow lines from the simulation. Figure 6.12 illustrates selected velocity flow frames from the pressure shock simulation. The water is constrained in a vertical capsule and a pressure shock can be replicated by shrinking the capsule in all dimensions. Due to

collision boundaries the water can only escape the high pressure increase by expanding at the free surface. In **a)** the capsule container has been shrunken and the water is building up a high pressure, which in **b)** consequently creates a water shock explosion that expands the water particles spherically. The water can only expand at the free surface as in **c)**, but it is interesting to verify that the particles within the water are actually sucked up with the surface expansion. In **d)** the surface turbulence develops into splashes, while the particles within the water start to fill up the volume again, e.g. seeking back from the compressed center. In **e)** the surface splashes are getting calmer due to gravity, and finally in **f)** the particle splashes return to the surface.

We have not compared the pressure shock results with any real experimental data on the matter, but we find it both interesting and convincing to how the intrinsic water particles react to the pressure shock.



a) $t = 0.06s$ **b)** $t = 0.13s$ **c)** $t = 0.22s$

d) $t = 0.27s$ **e)** $t = 0.37s$ **f)** $t = 0.45s$

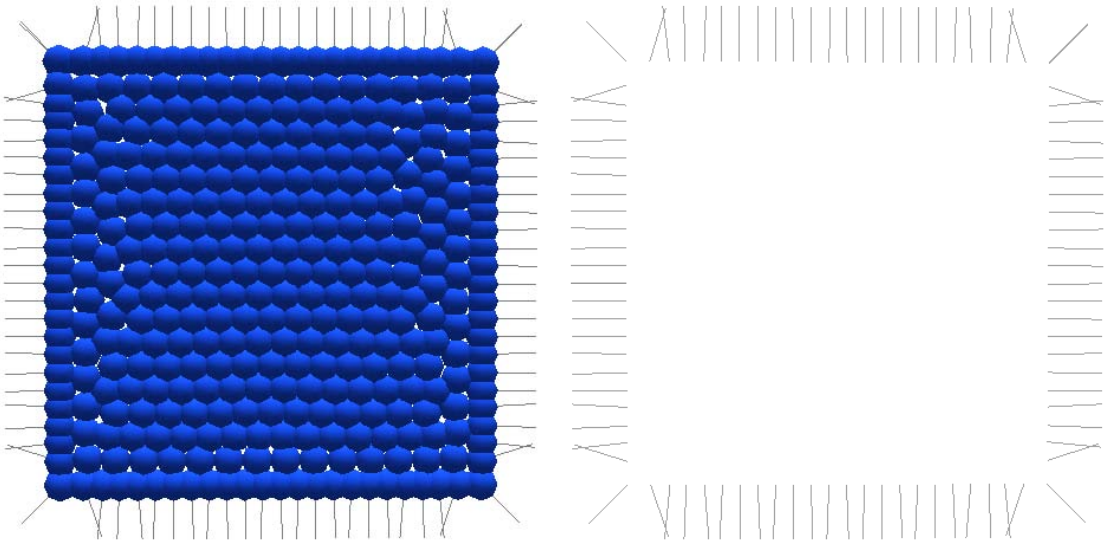
Figure 6.12 Water velocity flow from a vertical pressure shock. Exact simulation times are indicated below each frame.

6.3 Sanity and Stability

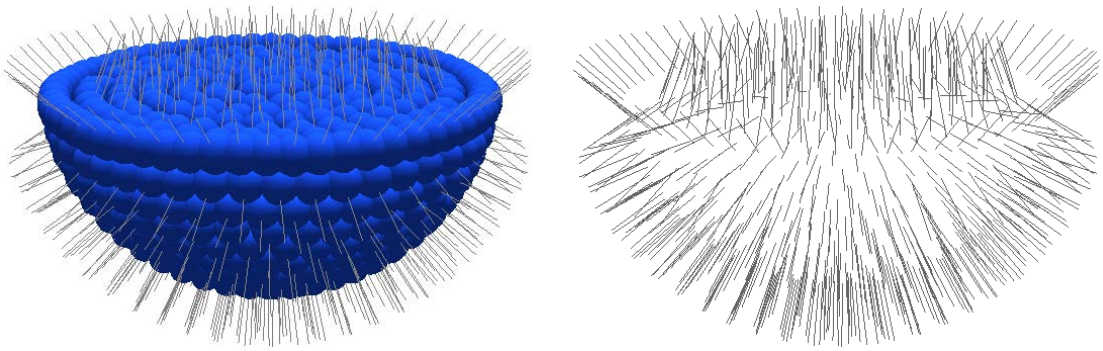
The effects of the physical parameters described in Section 5.3 are shown throughout this chapter. We believe that it is important to know that they are making sense for the simulations. In this section we will show that the questionable surface tension threshold parameter for determination of surface particles does in fact work convincingly for our solver, even though is does not have any physical meaning. We will also focus on stability for small-scale simulations.

Our experiments show that the surface tension threshold parameter (5.17) can be used to determine particles located near or on the surface. These particles are used in the computations of the surface tension force for liquid fluids. On Figure 6.13 water is depicted in one-, two-, and three-dimensional situations, along with detected surface normals. The surface normals are computed using (5.18), and scaled to match the particles. The situation for the mucus fluid is the same as for water.

a) Surface normals for 1-dimensional water.

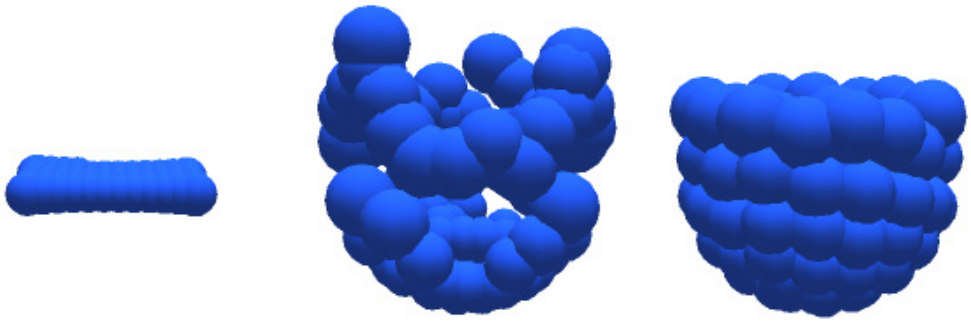


b) Surface normals for 2-dimensional water.

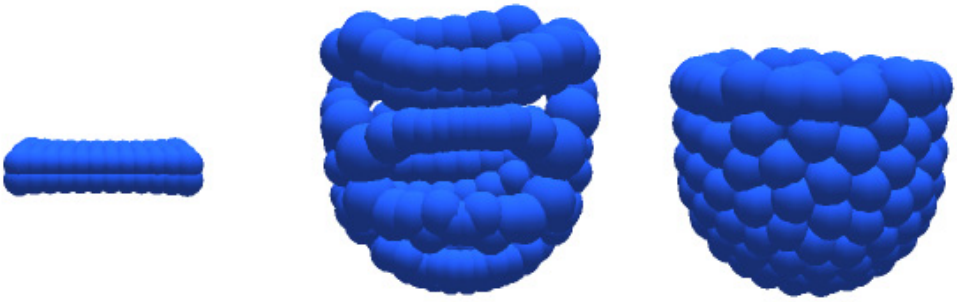


c) Surface normals for 3-dimensional water.

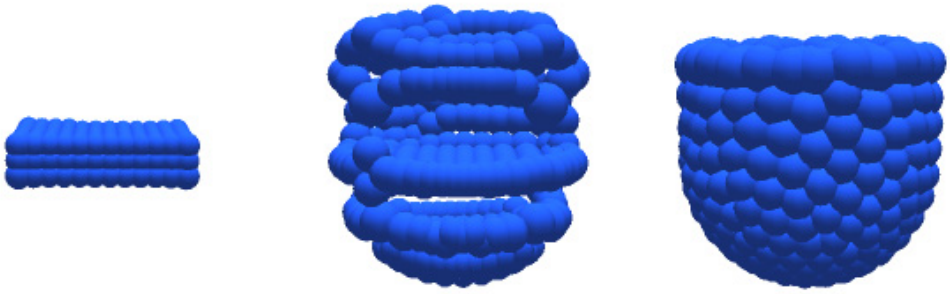
Figure 6.13 Surface normals with water particles to the left and without to the right. The water particles have been structured in **a)** 1D, **b)** 2D, and **c)** 3D.



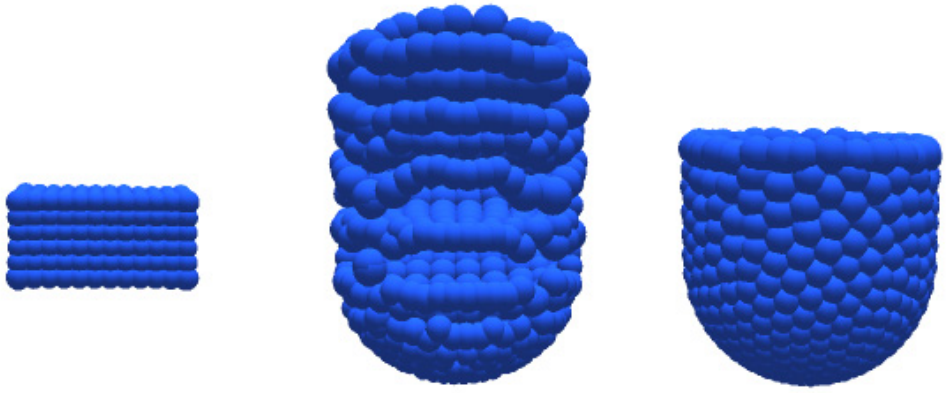
a) $0.1\ m^3$ water represented by 100 particles of $0.9983\ kg$ each.



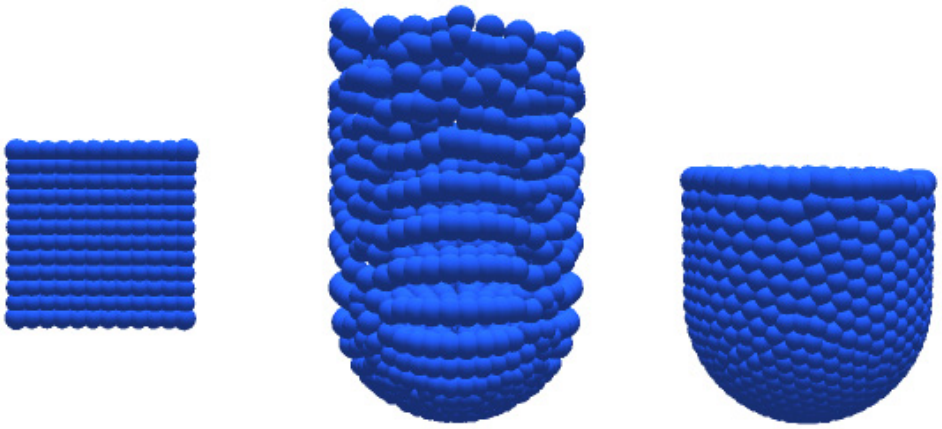
b) $0.1\ m^3$ water represented by 200 particles of $0.4991\ kg$ each.



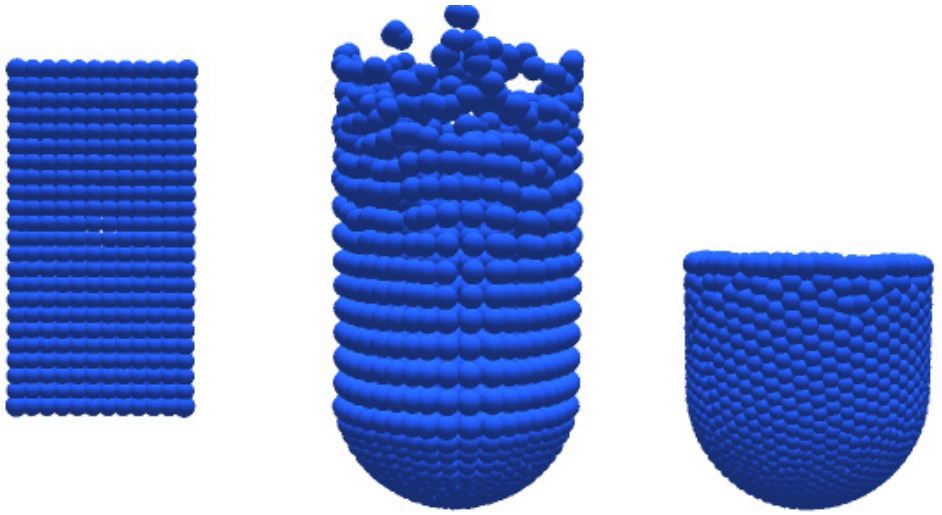
c) $0.1\ m^3$ water represented by 400 particles of $0.2496\ kg$ each.



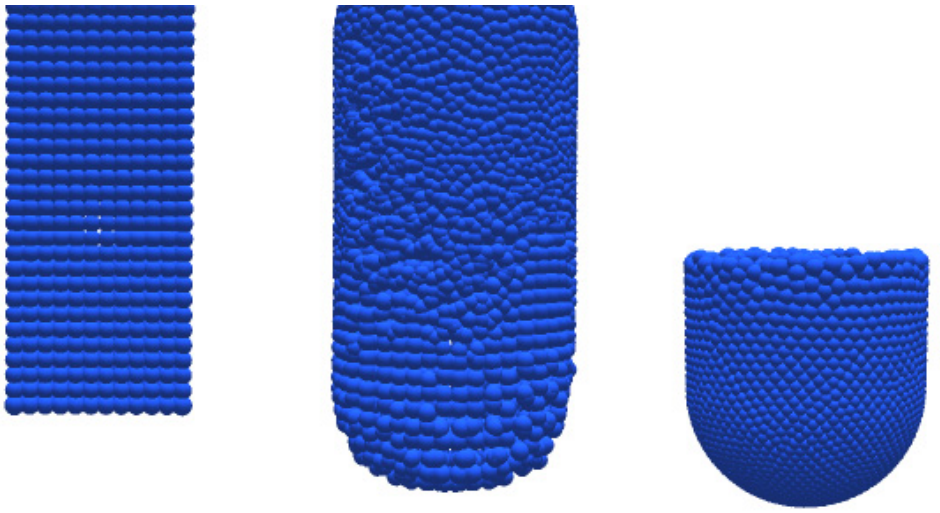
d) $0.1\ m^3$ water represented by 800 particles of $0.1248\ kg$ each.



e) $0.1\ m^3$ water represented by $1{,}600$ particles of $0.0624\ kg$ each.



f) $0.1\ m^3$ water represented by $3{,}200$ particles of $0.0312\ kg$ each.



g) $0.1\ m^3$ water represented by $6{,}400$ particles of $0.0156\ kg$ each.

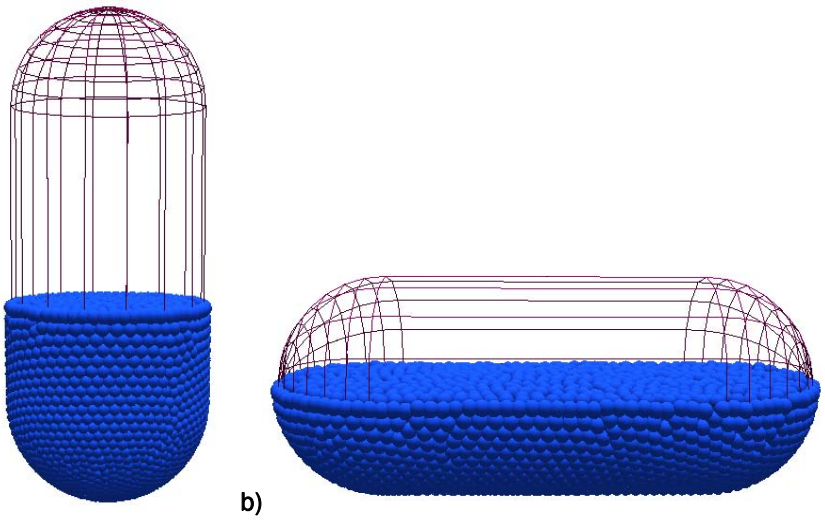
Figure 6.14 A vertical capsule filled with $0.1\ m^3$ water represented by different amounts of particles. The first column shows the initial particles, the second column shows the particles right after simulation start, and the third column shows the particles at near-rest.

The correlation between the kernel support radius, fluid volume, and particle mass is visualized on Figure 6.14. The figure shows that the fluid solver is stable using (5.12) and (5.14) for small-scale simulations. The third column depicts the water at near-rest, but due to incompressibility issues the visualized volume is not completely equal. In **g)** the initial particles reach beyond the capsule shell, which explains the high concentration of particles in the middle column.

6.4 Performance Tests

In this test we will monitor the performance frequency of the Lagrangian fluid application. The frequencies are measured in frames per second. Unfortunately, we do not have any performance references for other particle-based solvers, other than our own experience with RealFlow[3] from Section 1.1, but the performance tests will give us an idea of the overall interactivity with the application. The performance tests are executed on a standard PC with an AMD Athlon64 3400+ 2.4GHz processor with 1GB of memory running the Microsoft Windows XP operating system.

The interesting part is how many particles we can simulate while still being able to interact with the system. We will begin by using 100 particles, and continuously increasing the amount of particles by 100 until we reach $5{,}000$ particles. We will perform the tests on a single scene using a capsule as collision container. For each subtest the particles are initially positioned as a block inside the capsule with no initial velocity. The simulation is started and the particles will fall under gravity and collide with the inside of the capsule. The whole system must be relaxed before we continue to the next subtest, i.e. all particle velocities must be zero or very close to zero. Each second we record how many frames we have simulated and rendered since last record. Each subtest will be executed twice, with the capsule in a vertical position and in a horizontal position, as depicted on Figure 6.15.



a) **b)**

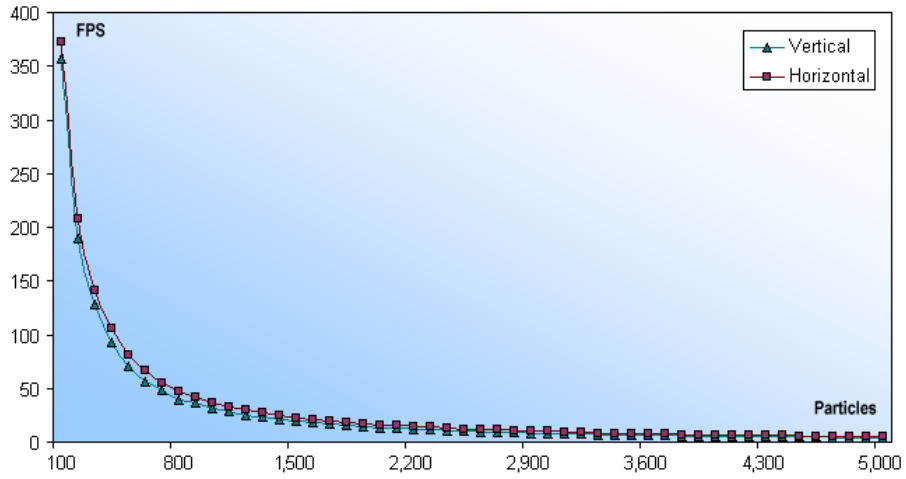
Figure 6.15 The two performance tests using the same capsule **a)** in vertical position and **b)** in horizontal position. $5{,}000$ particles are inside the capsule on the figures.

In Table 6.1 we have listed a selection of the performance measure results. On Figure 6.16 all recorded subtests results are plotted on a performance chart. The reason to why the frame rate measurements from the horizontal capsule are slightly better than the measurements from the vertical capsule is most likely implicitly caused by the lack of incompressibility. The pressure is higher in the bottom of the vertical capsule than in bottom of the horizontal capsule. High pressure means high mass-density, and high mass-densities means particles are crammed, thus more particles are within the compact support radius. The chart on Figure 6.16b is scaled logarithmically on the y-axis and from about $1{,}000$ particles the graphs are approximately linear, which is a common indication of cache miss problems.

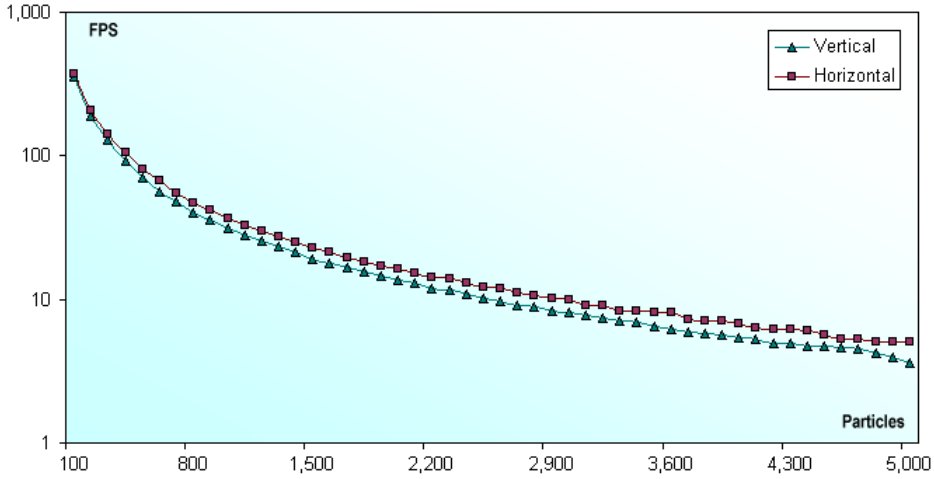
From Table 6.1 we can state that no more than $2{,}500$ to $3{,}000$ water particles can be simulated interactively in the current working version of our fluid solver. When the frame rate drops below $10\ fps$ the fluid will no longer react convincingly to the interactions performed by an end user. Disabling the spatial hashing grid for fast neighbor search, and thus performing the standard $O\left(n^2\right)$ operations for each particle, the frame rate quickly drops below $10\ fps$ if more than 750 particles are used in a simulation. This, however, proves that the spatial hashing search grid does improve the performance greatly. Still, we believe that performance can be improved further, and some of the performance issues can be dealt with as discussed in Section 5.7, while other issues might need a thorough analysis to determine the actual internal bottlenecks in the system.

Particles	Vertical $[fps]$	Horizontal $[fps]$
100	356.24	372.11
200	189.60	208.30
400	92.92	105.75
600	56.43	66.57
800	39.78	47.36
1,000	31.33	36.75
1,200	25.38	29.63
1,400	21.25	24.80
1,600	17.94	21.35
1,800	15.50	18.17
2,000	13.53	16.12
2,500	10.18	12.13
3,000	8.02	9.93
3,500	6.46	8.10
4,000	5.36	6.82
4,500	4.74	5.68
5,000	3.61	5.00

Table 6.1 Results from the performance tests.



a) Performance chart, normal scale y -axis.



b) Performance chart, logarithmic scale y -axis.

Figure 6.16 Performance charts.

One final note about Figure 6.15 is that it is worth noticing that even though the same amount of particles are simulated in both cases, the horizontal capsule is approximately half-full, while approximately only 40% is filling the vertical capsule. This is an issue regarding the lack of incompressibility, and is discussed in the next section.

6.5 Issues and Challenges

Incompressibility has been mentioned several times throughout the report. Section 5.2 states that incompressibility can be obtained when $k \to \infty$, which also indicates $\Delta t \to 0$. However, near-incompressibility can be obtained more painlessly by decreasing Δt by a factor of 10 and thus increasing the gas stiffness constant as much as possible within a stable range. We believe that near-incompressibility can be accepted in Computer Graphics, but a time step of only $1\ ms$ requires the simulation to be executed 10 times at each frame to obtain the same visual experience. Running the fluid simulation 10 times at each frame will cause the

application to deviate from any interactivity if more than $1,000$ particles are used in a scene. Figure 6.17 and Figure 6.18 depict the same water simulation but with different time step and gas stiffness constant. The frames from the two simulations are synchronized to match the exact simulation times. The water is expanding very rapidly in the first couple of milliseconds on Figure 6.18, compared to Figure 6.17. This is caused by the initial positions of the particles, which seems to be too compressed for a nearly incompressible fluid. Near-incompressibility is obtained when using $\Delta t = 0.001s$ and $k = 100J$, which can be verified on Figure 6.18. However, the water seems livelier on Figure 6.17.

Incompressibility is not only of visual importance. Numerical instabilities can occur in situations where a large quantity of compressible fluid is constrained in a pillar container, e.g. the pressure at the bottom becomes extremely high on a small area. On Figure 6.19 a standard water simulation of $4,400$ particles is illustrated. The water never calms down. Due to the high pressure at the bottom the water reacts very turbulently, and continuously emits particles along the faces of the pillar. We continued the simulation up to a simulation time of $t = 100\ s$, but there was no change in the unstable behavior. On Figure 6.20 the same water simulation as from Figure 6.19 is depicted, but this time we have used the settings for the near-incompressible water. No instabilities occur in this situation. It is clear that the water calms down starting at the bottom of the pillar.

The instabilities from Figure 6.19 do not occur the same way for smooth containers, e.g. a capsule. The implicit box is discontinuous at the edges and that can have an effect on the compression of the water particles. The instabilities are reduced greatly if the pillar is lying down, as the high pressure is distributed over a larger area.

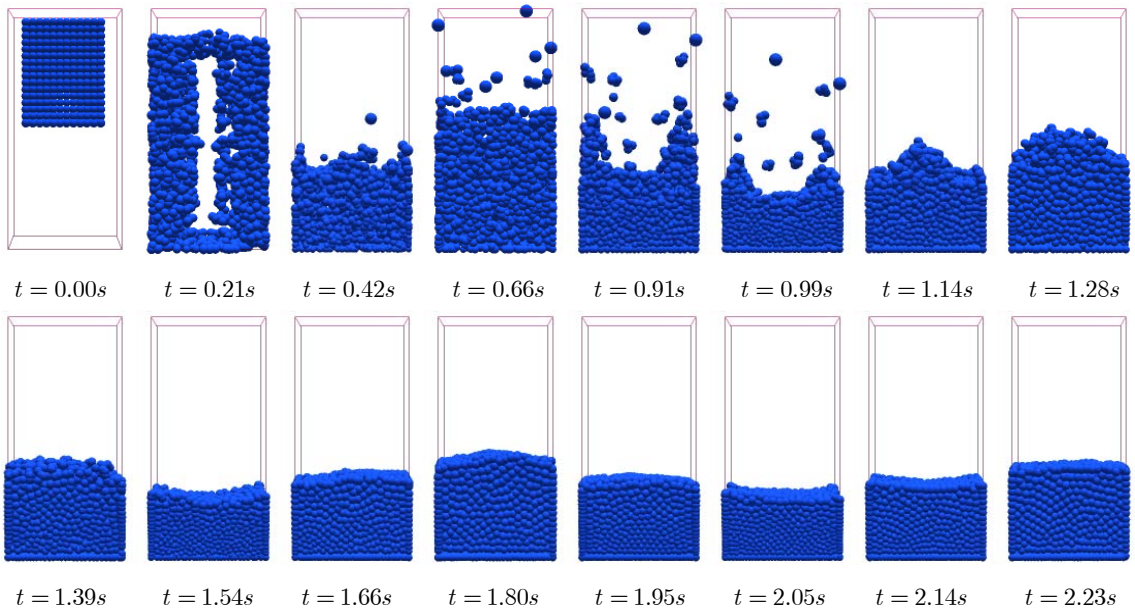


Figure 6.17 The fluid is too compressible for $0,045\ m^3$ water simulated using $2,250$ particles with $\Delta t = 0.01s$ and $k = 3J$. Simulation times are listed below the frames.

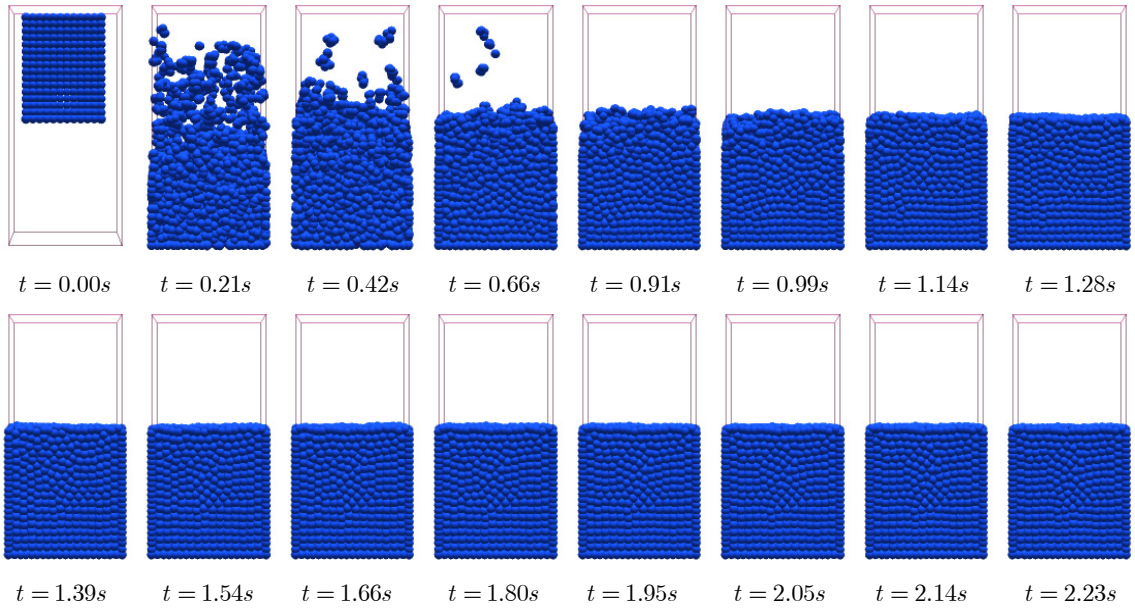


Figure 6.18 Near-incompressibility is obtained perfectly for $0,045\ m^3$ water simulated using $2,250$ particles with $\Delta t = 0.001s$ and $k = 100J$. Simulation times are listed below the frames.

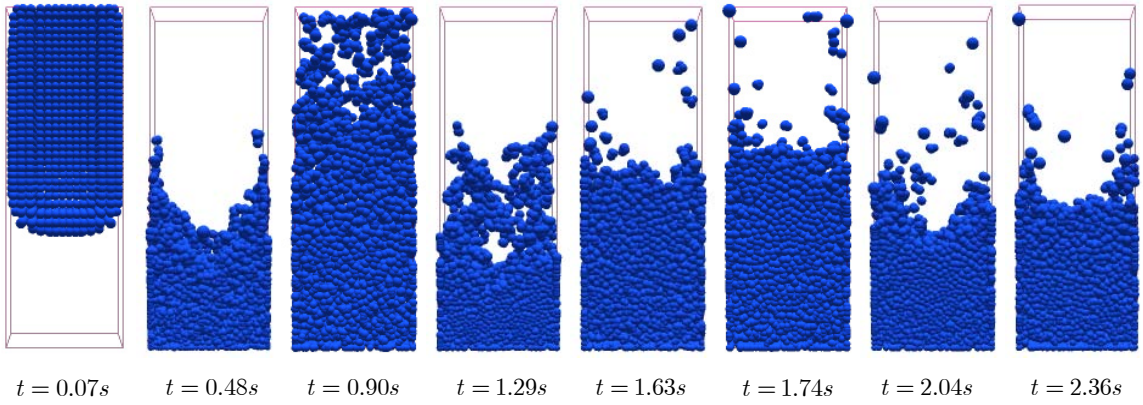


Figure 6.19 Instabilities occur when the pressure becomes too high for a compressible fluid. $0,088\ m^3$ water constrained in a pillar container. Exact simulation times are listed below the frames.

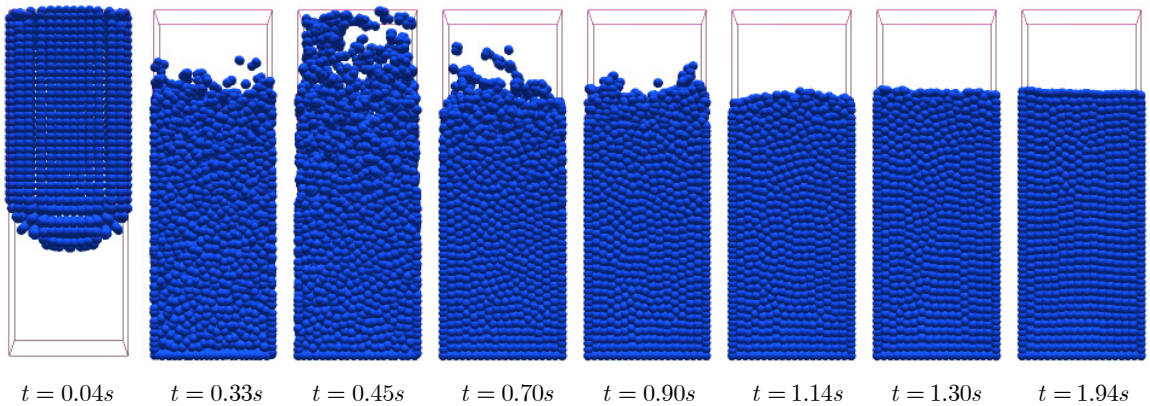


Figure 6.20 No instabilities occur for a near-incompressible fluid, as the pressure never becomes too high. Simulation of $0,088\ m^3$ water. Exact simulation times are listed below the frames.

The asymmetrical surface tension force density causes strange fluid motion if the free surfaces of a fluid are represented in one or two dimensions. Figure 6.21 depicts a little interesting simulation of the viscous mucus fluid. All particles are constrained at the bottom of the box, which subsequently is expanded. As illustrated on Figure 6.13b the surface tension force only works on the boundary of the mucus in two dimensions. In the ideal world the surface tension force would transform the free mucus into a circle, but instead it slowly but continuously transforms the mucus into every other two-dimensional smooth shape than a circle. As soon as the free surface of an isolated fluid lump is represented in three dimensions the surface tension force correctly transforms the fluid into a spherical shape, e.g. droplets.

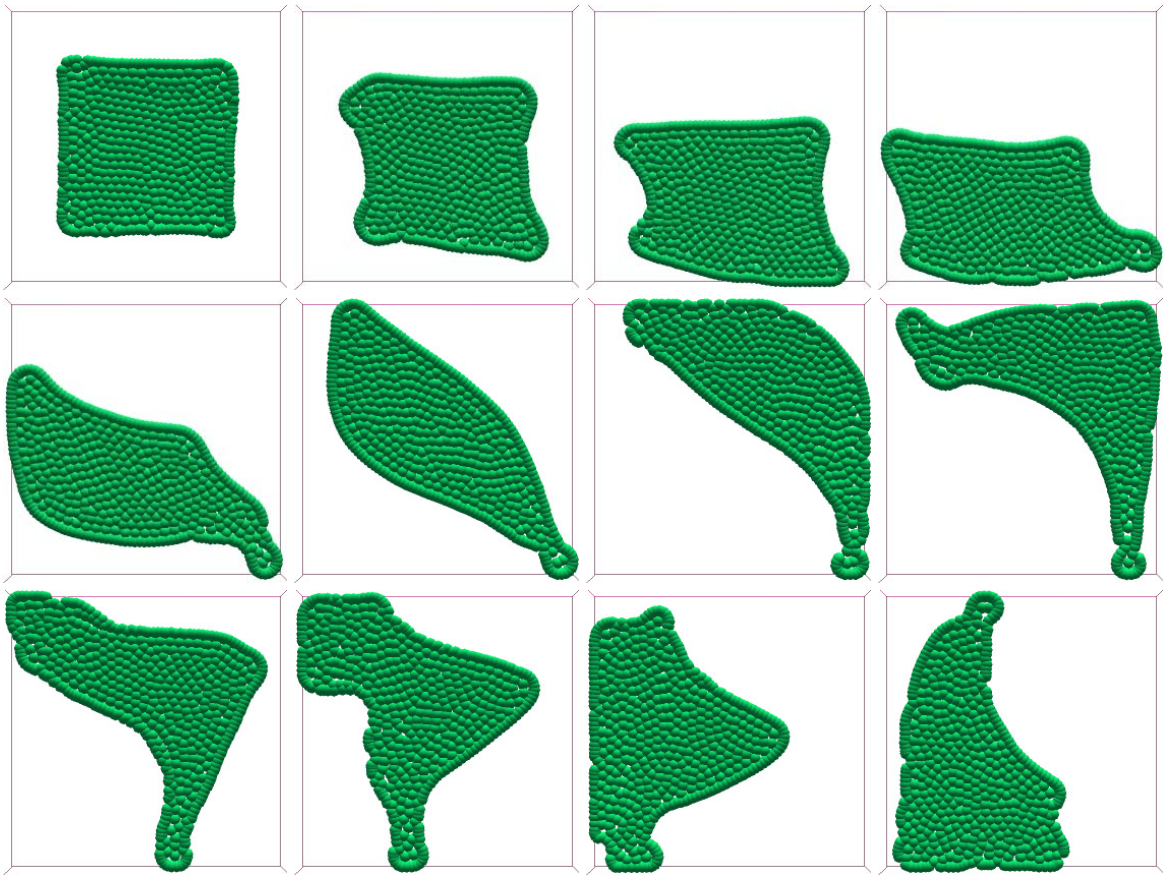


Figure 6.21 The asymmetrical surface tension force cause interesting but wrong fluid motion. The simulation time interval is $1s$ between each frame from left to right, top to bottom.

7 Future Work

The Lagrangian way of fluid simulation using smoothed particle hydrodynamics makes CFD in 3D possible at interactive rates. The methods presented in this work are fundamental and serve as the basic building blocks for interactive fluids. Although one way to increase the computational performance has been presented in Section 5.1, there is still room for improvements and extensions. In this chapter some of the more interesting improvements and extensions are presented, some already categorized as work in progress.

7.1 GPU Utilization

The graphics processing unit (GPU) has become the most common auxiliary tool to boost the performance in graphics and simulation applications. With the technology of a powerful pixel shader that can perform billions of floating point operations per second in parallel, it will make sense use a GPU in aid for the fluid SPH calculations. Force density computations, time integration, and visualization can all be performed on the GPU. If the three steps can be performed entirely on the GPU without transferring data back to the CPU for additional arrangements, maximum performance can be obtained. However, as this is not trivial to achieve, some efforts must be put into this research.

7.2 Fluid Visualization

In this work fluid particles are only visualized as spheres with individual radii. The possibility to examine the particles makes the fluid flow evident. This is interesting for an engineer or researcher, but might not impress the observer that expected to see water and other types of liquids.

One way to extend the visualization of the fluid particles is to use surface splatting [38], which is an image space technique that directly renders opaque and transparent surfaces from point clouds without connectivity. Simpler methods also exist if only the surface is of interest. The hardware accelerated method for point-based rendering of isosurfaces in [2] employs an OpenGL implementation that can render points very efficiently. We can use a similar technique to point render the particles that have already been identified as being on the fluid surface.

Another more traditional procedure is to render a reconstructed surface. Within the area of surface reconstruction lies surface representation. To find a representation of the fluid surface a function reconstruction method can be applied to construct an implicit function that best possible describes the surface of the fluid. The moving least-squares (MLS) method has an approach similar to the color field used in this work for identifying particles located on the surface. In [31] the MLS method is used to represent an implicit function from unorganized points and

polygons. However, our smoothed color field (4.27) already provides an implicit expression of the fluid surface.

Once the surface representation of the fluid has been obtained several techniques can be applied to extract and render the final fluid surface. A common high quality offline rendering method is Ray Tracing/Casting, which can be applied directly to the surface representation. For graphics applications a polygon mesh that approximates the fluid surface is often required, but must first be extracted from the surface representation. A very common method for surface extraction is the marching cubes algorithm [19] that can triangulate any isosurface from the implicit surface representation.

7.3 Incompressible Lagrangian Fluids

While the SPH method for Lagrangian fluids is fast and flexible, it only allows for simulation of compressible fluid flow. So far incompressibility for SPH has not been obtained satisfactory, as the particle mass-densities are not constant in general. In this work the level of compressibility in SPH is controlled by the gas stiffness constant k in the pressure spring (5.10). Incompressibility can be obtained by using infinitely stiff pressure springs. However, this will cause violent instability problems that might only be solved using an infinitely small time step. An interesting research area is to follow up on the incompressibility relaxation method presented in [4] and to find an explicit solution that can maintain the incompressibility for the SPH method, without compromising the interactivity or the physical representation.

7.4 Advanced Fluid Interactions

Simulating fluids using Lagrangian particles makes it interesting to go into details on similar Lagrangian methods to simulate rigid and deformable bodies. If fluids, rigid bodies, and deformable solids all can be derived from Lagrangian particles, intriguing fluids interactions can be simulated, e.g. solid boxes that plump into a pool of water, creating a wave that fracture the pool [24]. Phase transition is yet another solid-fluid application that has been achieved using the Lagrangian SPH approach [16]. The solid-fluid interactions presented in this work suffer from any friction. Virtual boundary particles fixed in a body frame can be used to achieve sticky liquid effects between fluids and solids, e.g. droplets stick to a straw pulled from a glass of water, and juice inherited the rotation from a rotating glass.

Interactive fluid-fluid interactions have become manageable using SPH. Compared to an Eulerian method, it seems like the Lagrangian approach is best suitable for the simulation of the complex fluid-fluid interactions. In [25] simulations of a lava lamp illustrate the interactions between different fluid substances, and by using different rest densities for different fluids, e.g. water and air, a buoyancy effect can be simulated, e.g. rising air bubbles in water.

8 Conclusion

In this report we have studied smoothed particle hydrodynamics, a method that when used rationally can render interactive fluid simulations possible without additional hardware support. We have derived the Navier-Stokes equations for particle-based fluid motions and employed the SPH method to compute all the complex physical quantity fields. Applying SPH directly to the different force densities does not always guarantee in conservation of Newton's 3rd law. We have used additional SPH techniques to symmetrize the internal forces such that linear and angular momenta are conserved.

The SPH method is a powerful tool that reduces the complexity of the mathematical equations of fluid flows, but as it originally was designed for compressible flow problems, the lack of incompressibility is the one major drawback of the method. A liquid fluid, like water, is generally considered incompressible for small-scale systems, and that assumption is further used in the derivation of the Navier-Stokes equations. We have shown how to use SPH for near-incompressibility without any redesign of the model. However, near-incompressibility implies using smaller integration time steps, which consequently affects the interactivity.

We have described the meaning of the physical parameters required to simulate particle-based fluids, and we have tried to find a connection between some of them in ways that can be considered intuitive for an end user. Further, we have found that the dependencies between the connected parameters work well with respect to stability, and allow for various types of fluids to be described without much work.

Based on the theory presented in this report we have implemented an open-source particle-based fluid solver that runs interactively for various types of small-scale simulations. However, even though up to $3,000$ particles can be simulated convincingly, the performance rate is not as good as we have hoped. We believe that a higher simulation frequency can be obtained once we improve the cache issues. Still, we have proven that Lagrangian fluid dynamics using SPH is superior to Eulerian fluid dynamics when it comes to real-time fluid simulations. This conclusion is drawn based on the computation times from Section 2.2 and our own performance measurement from Section 6.4.

We believe that we have achieved our goals of describing and developing a method to simulate fluids at interactive rates and to use as physically correct fluids parameters as possible. Although, we have not focused on the visualization of the free surfaces, a fast and visually pleasing method is required for the SPH method to become useful in e.g. computer games. With our work in progress on real-time fluid visualization we further believe that our particle-based fluid solver will be an acknowledged contribution to OpenTissue that can inspire potential users of interactive fluid modeling and design.

8.1 Contributions

Based on the previous work done in the field of particle-based fluid simulations, this report contributes:

- A thorough insight of the mathematical theory of particle-based fluid motion
- Physically correct fluid parameters
- Stable collision handling between fluid particles and implicit primitives
- Visual analysis of fluid flows using SPH in Computer Graphics
- A complete open-source implementation of particle-based fluids

References

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. "*Computational Geometry: Algorithms and Applications*", second edition. Springer-Verlag, 2000.

[2] J. A. Bærentzen and N. J. Christensen. "Hardware Accelerated Point Rendering of Isosurfaces". *Journal of WSCG*, vol. 11, no.1, pp. 41-48, 2003.

[3] A. Colagrossi and M. Landrini. "Numerical simulation of interfacial flows by smoothed particle hydrodynamics". *Journal of Computational Physics*, Volume 191, Issue 2, pp. 448-475, 2003.

[4] S. Clavet, P. Beaudoin, and P. Poulin. "Particle-based Viscoelastic Fluid Simulation". *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pp. 219-228, 2005.

[5] M. Desbrun and M.-P. Cani. "Smoothed Particles: A new paradigm for animating highly deformable bodies". *In Computer Animation and Simulation '96*, pp. 61–76, 1996.

[6] D. Eberly. "*Game Physics*". Morgan Kaufmann, 2003.

[7] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann. *"Physics-Based Animation"*. Charles River Media, 2005.

[8] N. Foster and R. Fedkiw. "Practical Animation of Liquids". *In proceedings of SIGGRAPH 2001*, pp. 15-22, 2001.

[9] N. Foster and D. Metaxas. "Modeling the Motion of a Hot, Turbulent Gas". *In Computer Graphics Proceedings 1997*, Annual Conference Series, pp. 181–188, 1997.

[10] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien. "A Method for Animating Viscoelastic Fluids". *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004)*, vol. 23, pp. 463-467, 2004.

[11] M. J. Harris. "Fast Fluid Dynamics Simulations on the GPU". *In GPU Gems, Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Edited by R. Fernando, Chapter 38. Addision-Wesley, 2004.

[12] J. Hongbin and D. Xin. "On criterions for smoothed particle hydrodynamics kernels in stable field". *Journal of Computational Physics*, 202, pp. 699–709, 2005.

[13] B. Houston, M. Wiebe, and C. Batty. "RLE Sparse Level Sets". *Proceedings of the SIGGRAPH 2004 Conference on Sketches & Applications*, 2004.

[14] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. "Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation". *To appear in ACM Transactions on Graphics, 2006*. Conditionally Accepted April 4, 2005.

[15] T. Jakobsen. "Advanced Character Physics". *In proceedings of Game Developer's Conference*, 2001.

[16] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutré, and M. Gross. "A Unified Lagrangian Approach to Solid-Fluid Animation". *Proceedings of the Eurographics Symposium on Point-Based Graphics*, 2005.

[17] S. Koshizuka, H. Tamako, and Y. Oka. "A particle method for incompressible viscous flow with fluid fragmentation". *Computational Fluid Dynamics Journal*, 4, pp. 29-46, 1995.

[18] A. T. Layton and M. van de Panne. "A Numerically Efficient and Stable Algorithm for Animating Water Waves". *The Visual Computer*, Vol. 18, No. 1, pp. 41-53, 2002.

[19] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm". *In Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 163–169, 1987.

[20] F. Losasso, F. Gibou, R. Fedkiw. "Simulating Water and Smoke with an Octree Data Structure". *In proceedings of SIGGRAPH 2004*, pp. 457-462, 2004.

[21] J. J. Monaghan. "Smoothed Particle Hydrodynamics". *Annual Review of Astronomy and Astrophysics*, 30, pp. 543-574, 1992.

[22] M. Moore and J. Wilhelms. "Collision Detection and Response for Computer Animation". *In Computer Graphics*, Volume 22, pp. 289-298, 1988.

[23] M. Müller, D. Charypar, and M. Gross. "Particle-Based Fluid Simulation for Interactive Applications". *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation*, pp. 154-159, 2003.

[24] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. "Interaction of Fluids with Deformable Solids". *In Journal of Computer Animation and Virtual Worlds (CAVW)*, vol 15, no. 3-4, pp. 159-171, 2004.

[25] M. Müller, B. Solenthaler, R. Keiser, and M. Gross. "Particle-Based Fluid-Fluid Interaction". *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 237-244, 2005.

[26] M. B. Nielsen and K. Museth. "Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets". *Journal of Scientific Computing*, 2005.

[27] S. Osher and R. Fedkiw. "*Level Set Methods and Dynamic Implicit Surfaces*". Vol. 153 of Applied Mathematical Sciences. Springer, 2003.

[28] OpenTissue. "*Opensource Project, Physical based Animation and Surgery Simulation*". 2005. www.opentissue.org.

[29] PhysX. "*PhysX*". AGEIA. 2005. http://www.ageia.com/products/physx.html.

[30] RealFlow3. "*RealFlow³*". Next Limit Technologies. 2005. http://www.nextlimit.com/realflow/index.html.

[31] C. Shen, J. F. O'Brien, J. R. Shewchuk. "Interpolating and Approximating Implicit Surfaces from Polygon Soup". *The Proceedings of ACM SIGGRAPH 2004*, pp. 896-904, 2004.

[32] J. Stam and E. Fiume. "Depicting Fire and other Gaseous Phenomena using Diffusion Processes". *Computer Graphics*, 29th Annual Conference Series, pp. 129–136, 1995.

[33] J. Stam. "Stable Fluids". *In Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121-128, 1999.

[34] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret, and J.-D. Gascuel. "Animating Lava Flows". *In Graphics Interface*, pp. 203–210, 1999.

[35] D. Terzopoulos, J. C. Platt, A. H. Barr, and K. Fleischer (1987), "Elastically deformable models", *Computer Graphics,* volume 21, Number 4, July 1987, pp 205-214, 1987.

[36] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, and M. Gross. "Optimized Spatial Hashing for Collision Detection of Deformable Objects". *In proceedings of Vision, Modeling, Visualization*, pp. 47-54, November 19-21, 2003.

[37] L. Verlet. "Computer Experiments on Classical Fluids I: Thermodynamical of Lennard-Jones Molecules". *Physics Review*, vol. 159, pp. 98-103, 1967.

[38] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. "Surface Splatting". *In Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 371–378, 2001.