

# **Physically based Animation of Free Surface Flows with the Lattice Boltzmann Method**

## **Physikalische Animation von Strömungen mit freien Oberflächen mit der Lattice-Boltzmann-Methode**

Der Technischen Fakultät der Universität Erlangen-Nürnberg,  
zur Erlangung des Grades:

*Doktor-Ingenieur*

Vorgelegt von:  
Dipl. Inf. Nils Thürey

Erlangen, 2007

Als Dissertation genehmigt von  
der Technischen Fakultät der  
Universität Erlangen-Nürnberg

Tag der Einreichung:	2006-10-09
Tag der Promotion:	2007-03-13
Dekan:	Prof. Dr. Winnacker
Prüfungskollegium:	Prof. Dr. U. Rüde Prof. Dr. M. Pauly Dr. M. Breuer Prof. Dr. M. Stamminger

# Abstract

The numerical simulation of fluids has become an established tool in many engineering applications. Free surface fluids represent a special case that is important for a variety of applications. For a free surface simulation, a two phase system, such as air and water, is described by a single fluid phase with a sharp interface and corresponding boundary conditions. This allows the efficient representation and simulation of complex problems. In this thesis, the main application for free surface flows will be the generation of animations of liquids. Additionally, engineering applications from material science and particle technology are considered.

The simulation algorithm of this thesis is based on the lattice Boltzmann method. This method has been chosen due to the overall computational efficiency of the basic lattice Boltzmann algorithm, and its ability to deal with complex geometries and topologies. The basic algorithm is extended to compute the motion of free surfaces in three dimensions while conserving the overall mass. Adaptive time steps and grids, in combination with a turbulence model, allow stable and efficient simulations of detailed fluids. In combination with boundary conditions for moving and deforming objects, the algorithm represents a flexible basis for free surface simulations. Its performance on single CPU machines will be evaluated. Likewise, performance results of parallelized versions will be given for shared- and distributed-memory architectures.

For fluids in computer generated animations it is important to give animators control of the fluid motion. An approach to perform this fluid control, without disturbing the natural fluid behavior, will be given. Another typical problem is the simulation of large open water surfaces due to the highly differing scales of waves and drops. An explanation of how to perform such simulations using a combination of two-dimensional and three-dimensional techniques will be given, in combination with a particle based drop model. To demonstrate the capabilities of the solver that was developed during the work on this thesis, it has been integrated into an open source 3D application.

Finally, areas of future work and possible extensions of the algorithm will be discussed. One of these topics is the inclusion of an accurate and efficient curvature computation for surface tension forces. Furthermore, an outlook of possible applications in the fields of metal foams and colloidal dispersions will be discussed.



# Acknowledgements

First of all I would like to thank my supervisor, Ulrich Rüde, for the many helpful discussions and the ongoing support – even when my research drifted away from engineering applications towards more visually oriented topics. I am also grateful to the people who helped me writing this thesis by proofreading parts of it: Christian Feichtinger, Bettina Frohnappel, Jan Götz, Klaus Iglberger, Thomas Pohl, Stefan Thürey and Ben Bergen. Especially Ben removed a significant amount of "german-english".

Furthermore, numerous other persons were very helpful during the course of my work on this thesis. The many discussions with my colleagues Thomas Pohl, Jan Treibig, Christian Feichtinger and Klaus Iglberger were certainly valuable in many ways. Likewise Marc Stamminger, Thomas Zeiser, Frank Firsching, Anja Borsdorf, Günther Greiner, Quirin Meyer and Vivek Buwa helped by giving me vital hints for certain aspects of this work. Furthermore, Carolin Körner and Michael Thies were very important by developing the foundations of the LBM free surface algorithm, and helping me during my master thesis. Together with Thomas Pohl, who agreed to supervise my master thesis in 2003, they got me started with LBM and the free surfaces. Overall, many thanks to my colleagues at the LSS, who, among other things, helped me getting distracted from all the bugs and instabilities, e.g., by accompanying me to the Havanna Bar and the E-Werk. Kudos especially to those that did not use my Müsli-milk for their coffee.

Aside from my work at the LSS, the collaboration with ETH Zurich allowed me to work on one of the most interesting topics of this thesis – the control framework of Section 9. Mark Pauly and Richard Keiser significantly contributed to this work, while Markus Gross and Marc Stamminger helped to make this collaboration possible. Thanks also to the rest of the AGG & CGL teams for making it a very enjoyable visit.

In addition, the Blender development and user community was helpful testing the solver. Some people produced several impressive animations with it. Thanks also to Bassam Kurdali for providing the rigged character that was used in several test animations throughout the work on this thesis.

Finally and naturally, I want to thank the rest of my family – my parents Verena & Stefan, my brother Arne, my sister Jana, my grandparents Hannelore, Georg and Magdalena – for their encouraging comments in the last three years. They had to endure a stream of emails with strange fluid test pictures and buggy animations (that probably won't stop in the near future).

This work has been supported by the Graduate College GRK-244, *3-D Image Analysis and Synthesis*, of the German Science Foundation (DFG).

# Nomenclature

Note: Especially for parametrization and derivation of the simulation algorithm, all physical values will be marked with an apostrophe. The majority of the following equations, however, will use dimensionless lattice quantities. In the following table, values without units are by default non-dimensional, units are only given for physical quantities.

$f_i$	particle distribution function along an arbitrary velocity vector
$\tilde{f}_i$	distribution function along the inverse velocity vector of $f_i$
$f_i^{eq}$	equilibrium distribution function
$f_i^*$	post collision distribution function
$g_i$	particle distribution function for a shallow water LBM
$w_i$	LB equilibrium weighting factor
$m(\mathbf{x})$	fluid mass of the cell at position $\mathbf{x}$
$\epsilon(\mathbf{x})$	fill fraction of the cell at position $\mathbf{x}$
$C$	Smagorinsky constant
$P$	fluid pressure [ $\text{N}/\text{m}^2$ ]
$r$	domain grid resolution
$S'$	real world domain size [m]
$T$	fluid temperature [ $^\circ\text{C}$ ]
$\rho$	fluid density [ $\text{kg}/\text{m}^3$ ]
$\Delta t'$	physical time step [s]
$\Delta x'$	physical cell size [m]
$\mu'$	dynamic viscosity [ $\text{m Pa/s}$ ]
$\nu'$	kinematic viscosity [ $\text{m}^2/\text{s}$ ]
$\lambda$	relaxation time [s]
$\Delta t$	lattice time step
$\Delta x$	lattice cell size
$\nu$	lattice viscosity
$\tau$	lattice relaxation time
$\mathcal{E}$	fluid fraction deviation for accuracy measurements
$\mathbf{e}_i$	velocity vector of the LB model
$\mathbf{g}$	gravity acceleration vector $(0, 0, -9.81)^T$ [ $\text{m}/\text{s}^2$ ]
$\mathbf{u}$	fluid velocity [m/s]
$\mathbf{n}$	surface normal vector
$\mathbf{u}_o$	obstacle object velocity [m/s]
$\mathbf{n}_o$	obstacle object surface normal
$R$	Boltzmann constant $1.380650 \cdot 10^{-23}$ [ $\text{m}^2 \text{ kg s}^{-2} \text{ K}^{-1}$ ]
$Kn$	Knudsen number (ratio of mean free path and char. scale)
$Re$	Reynolds number
$\gamma$	energy, second hydrodynamic moment
$\varepsilon$	expansion parameter given by $Kn$

Additional notation for the Lagrangian drop model (Section 10.3):

$m_P$	drop mass [kg]
$\mathbf{w}$	drop velocity [m/s]
$\mathbf{w}_{rel}$	drop velocity relative to surrounding fluid [m/s]
$F_D$	drag force [N]
$F_G$	gravitational force [N]
$C_D$	drag coefficient

## Abbreviations

API	application programming interface
CAD	computer aided design
D2Q9	two-dimensional LB model with nine velocities
D3Q19	three-dimensional LB model with nineteen velocities
DF	distribution function (usually denoted $f_i$ )
BGK	Bhatnagar Gross Krook approximation of the collision operator
GUI	graphical user interface
LB	lattice Boltzmann
LBM	lattice Boltzmann method
LES	large eddy simulation
MRT	multi relaxation time model
MPI	distributed memory parallel programming API
NS	Navier-Stokes
OpenMP	shared memory parallel programming API
SPH	smoothed particle hydrodynamics
SWS	shallow water simulation
VOF	volume of fluid free surface simulation model



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Simulation of Free Surface Flows</b>	<b>5</b>
2.1	Animating Free Surfaces . . . . .	5
2.2	Comparing Simulation Approaches . . . . .	7
<b>3</b>	<b>The Lattice Boltzmann Method</b>	<b>11</b>
3.1	Historical Development . . . . .	11
3.2	The Basic Algorithm . . . . .	12
3.3	Stability . . . . .	16
3.4	Parametrization . . . . .	17
3.5	Derivation . . . . .	18
3.5.1	The Navier-Stokes Equations . . . . .	18
3.5.2	The Boltzmann Equation . . . . .	20
3.5.3	Chapman-Enskog Expansion . . . . .	21
3.5.4	Derivation of the Lattice Boltzmann Equation . . . . .	22
3.6	Closure . . . . .	26
<b>4</b>	<b>Lattice Boltzmann Simulations with a Free Surface</b>	<b>27</b>
4.1	Interface Movement . . . . .	29
4.2	Free Surface Boundary Conditions . . . . .	30
4.3	Flag Re-initialization . . . . .	31
4.4	Interface Cell Artifacts . . . . .	33
4.5	Interactive Simulations . . . . .	33
<b>5</b>	<b>Moving Obstacles</b>	<b>37</b>
5.1	Obstacle Boundary Conditions . . . . .	38
5.2	Moving Boundary Conditions . . . . .	38
5.3	Lattice Initialization . . . . .	39
5.4	Surface Generation . . . . .	42
5.5	Results . . . . .	44

<b>6 Adaptive Time Steps</b>	<b>47</b>
6.1 Adaptive Parametrizations and Mach Numbers . . . . .	47
6.2 Validation and Performance . . . . .	50
<b>7 Adaptive Grids</b>	<b>53</b>
7.1 Grid Refinement . . . . .	53
7.2 Adaptive Coarsening Algorithm . . . . .	55
7.3 Validation . . . . .	61
7.4 Performance . . . . .	64
<b>8 Parallelization</b>	<b>71</b>
8.1 OpenMP Parallelization . . . . .	71
8.2 MPI Parallelization . . . . .	74
<b>9 Fluid Control</b>	<b>79</b>
9.1 Generating Control Particles . . . . .	82
9.2 Control Forces . . . . .	82
9.3 Detail-Preserving Control . . . . .	84
9.4 Results . . . . .	85
<b>10 Modelling Large Scale Fluids</b>	<b>89</b>
10.1 Shallow Water Simulation . . . . .	90
10.2 Hybrid 2D/3D Simulation . . . . .	92
10.3 Lagrangian Drop Model . . . . .	96
10.4 Results and Discussion . . . . .	99
<b>11 A Programming Interface for Fluid Solvers</b>	<b>101</b>
11.1 Blender Integration . . . . .	101
11.2 Integration Extensions . . . . .	103
<b>12 Conclusions</b>	<b>107</b>
12.1 Summary . . . . .	107
12.2 Discussion and Future Work . . . . .	108
<b>A German Parts</b>	<b>129</b>
A.1 Inhaltsverzeichnis . . . . .	129
A.2 Zusammenfassung . . . . .	130
A.3 Einleitung . . . . .	131
<b>B Curriculum Vitae</b>	<b>133</b>

A complex, abstract visualization of a liquid simulation. The image shows intricate, swirling patterns of light blue, white, and green, representing the flow and turbulence of a liquid. The patterns are highly detailed, showing the internal structures and boundaries of the simulated fluid.

Water is the first Principle of Nature.  
*Thales of Milet, 546 BC*

# Chapter 1

## Introduction

Liquid phases of matter are vital for numerous events in nature. As such, they are part of every day life as well as many production processes. The numerical simulation of liquid and gaseous materials has become an important tool, e.g., to compute a weather forecast or the optimal shape of an aeroplane. This thesis will focus on the numerical simulation of problems where two phases are involved, such as water and air, and the gas phase can be handled with a simplified treatment. This is known as the free surface approach. In Figure 1.1 and 1.2, several examples of flows in nature with different scales can be seen, all of which could be theoretically recreated by a free surface simulator. Although they represent a special case of fluid simulation, free surface simulations are still valid for a wide variety of problems, from casting and foaming applications, to the animation of liquids for special effects in computer generated animations.

Tools for physically based animations are currently included in all major 3D applications, as many physical effects are not suited for traditional animation approaches, such as keyframing. Physically based animation means that the laws of physics are solved or approximated with numerical algorithms to automatically create realistic behavior and plausible motion of animated objects. Typical examples are collapsing stacks of boxes, or the motion of clothes of an animated character. Algorithms for physically based animations often represent a combination of traditional numerical simulations, e.g., for civil and engineering purposes, and animation algorithms for computer graphics. While engineering applications usually focus on physical accuracy, a realistic appearance is the most important aspect for computer generated animations. In

---



Figure 1.1: Small scale example of a real fluid that could be simulated with a free surface model – filling a cup with tea.

both cases, however, efficiency of the applied algorithms is highly important for practical applications. As fluids are so common in nature, it is furthermore crucial for a plausible appearance that, e.g., splashes of water do not pass through each other, change their path in mid-air, or disappear during their movement. Moreover, the most common fluid, water, has a viscosity close to zero. Thus, it exhibits a very turbulent behavior, which has to be recreated for computer animations and which is very hard to recreate without relying on numerical simulations.

The physically based animation of liquids relies on the same underlying mathematical descriptions, the *Navier-Stokes* (NS) equations, which have traditionally been used in numerical simulations for a long time. The first two-dimensional computations of flows around cylinders were performed around 1930, while three-dimensional problems were not solved until around 1965 [HS66]. Furthermore, standard free surface simulations have constraints similar to those that are important for a realistic appearance – e.g., mass conservation and turbulence. Therefore, this thesis will explain how to extend the free surface simulation algorithm developed for the simulation of metal foaming processes to efficiently create physically based animations of three-dimensional free surface flows. The algorithm is based on the *lattice Boltzmann method* (LBM), a relatively new approach to approximate the Navier-Stokes equations that has become increasingly popular due to its simple and efficient basic algorithm.

## Contributions of this Thesis

The goal of this thesis is the stable, efficient and flexible simulation of free surface flows with the LBM. The main contributions of this thesis to achieve this are:

- An algorithm for the treatment of moving and deforming obstacle objects to enable the simulation of dynamic and realistic scenes.
- Adaptive time steps to efficiently handle and stabilize scenes with varying velocities.
- Adaptive grids for the simulation of large water volumes. These can speed up detailed free surface simulations by more than a factor of three.
- A control mechanism that preserves small scale flow features. It gives animators large scale control of the fluid without disturbing the natural motion.
- A hybrid algorithm to enable simulations of large scale open water scenes by coupling two-dimensional and three-dimensional techniques in combination with a particle based drop model.

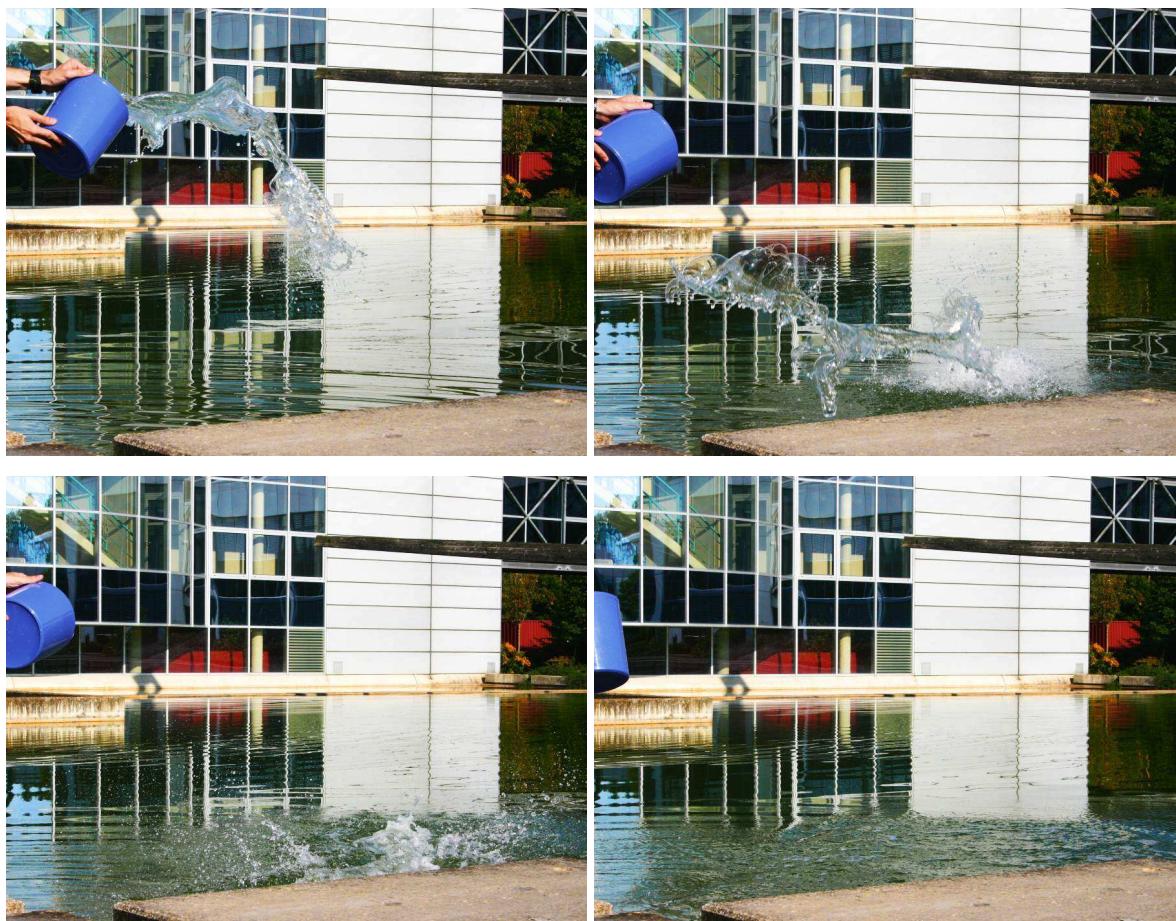


Figure 1.2: Here images from another real world free surface flow can be seen. Water is poured onto an open water surface.

---



## Chapter 2

# Simulation of Free Surface Flows

## 2.1 Animating Free Surfaces

This section will give a brief overview of related work on fluid simulations for computer animations. Further references can be found in the corresponding sections of each chapter. A first numerical fluid simulation was used in [YUM86] for generating the textures of the atmosphere of Jupiter, while Foster and Metaxas [FM96] were the first to perform physically based animations of free surface fluids. They applied an iterative scheme to solve a finite difference discretization of the NS equations, in combination with marker particles for the free surface. However, at the time, the algorithms and computational resources were not adequate to perform believable large scale animations. Jos Stam's introduction of projection methods led to the so-called semi-Lagrangian fluid solvers [Sta99]. It was an important step towards practical fluid animations as it significantly reduced the required computational resources.

After Ron Fedkiw's work on level set and fluid simulation algorithms, e.g., in [FAMO99], Foster and Fedkiw revived the topic of free surface animations five years after [FM96] in [FF01]. Together with [Sta99] this work represents the foundation of the now established group of level set based free surface simulations. These methods have since been extended in various ways, e.g., to enhance the tracking of the free surface [EMF02], to perform simulations on octree data structures [LGF04], or to handle coupled simulations with deformable shells [GSLF05]. These algorithms have the

---



Figure 2.1: A sample of a metal foam, provided by C. Körner from WTM Erlangen is shown on the left. An image of a foaming simulation from [KTH<sup>+</sup>05] performed with the free surface algorithm of Section 4 can be seen on the right side.

advantage of a continuous and smooth surface representation given by the level set. However, the level sets are not mass conserving by themselves. Thus, additional work has to be done to ensure mass conservation, e.g., by tracing additional particles near the interface [ELF05]. The method was furthermore extended to handle discontinuities at the interface in [HK05], to accurately simulate drops on surfaces [WMT05], and to enhance visual detail with vortex particles [SRF05]. [BGOS06] and [KAK<sup>+</sup>06] moreover present techniques to texture fluid surfaces without undesired diffusion effects.

A variety of other fluid simulation algorithms have moreover been used to create physically based animations. One class of these algorithms are the *Volume of Fluid* (VOF) methods [HN81, SZ99]. They track the free surface by computing the fraction of each volume of a cell in the computational grid that is filled with fluid. The advantage of these algorithms is the conservation of mass. On the other hand, additional work has to be done to create a fluid surface without artifacts or artificial surface tension. Sussman [Sus03] uses a VOF method combined with a level set, to achieve both mass conservation and a smooth surface representation. The algorithm is used to simulate breaking waves in [MMS04]. Apart from VOF, another popular class of methods are *smoothed particle hydrodynamics* (SPH). These stem from the field of astrophysics [Mon05], and have the advantage of working without the explicit need for a domain grid. The fluid is tracked by particles, which are also used to evaluate the computational kernels. Among others, the method is thus attractive in combination with a point based geometry framework [PKKG03, PPG04, AA06]. SPH has been successfully applied to simulate phase changes [MKN<sup>+</sup>04] or multi-phase fluids [KAG<sup>+</sup>05, MSRG05]. However, the repeated averaging due to the kernel evaluations can lead to smoothing effects, like an increased numerical viscosity. Unless multi resolution methods (e.g., as those described in [KAG<sup>+</sup>06]) are used, larger fluid particle numbers can furthermore lead to high computational costs due to the neighborhood computations.

More recently, variants of the semi-Lagrangian method of [Sta99] and alternative approaches have been proposed. One of these is the so called *FLIP* method [BR86], that is for example used to simulate sand as a fluid in [ZB05]. Another algorithm that works on arbitrary simplicial meshes is described in [ETK<sup>+</sup>06]. Its goal is to reduce the

inherent artificial viscosity and diffusion of a semi-Lagrangian solver using a vorticity formulation of the NS equations. [FOK05] and [KFCO06] demonstrate a discretization of the semi-Lagrangian solver for dynamically changing tetrahedral meshes. A different but similarly interesting approach is explained in [TLP06]. The authors use a model reduction technique to allow realtime simulations of phenomena such as smoke and fire.

The algorithm applied in this thesis – the LBM – has become popular in previous years, and is now used for a variety of applications. Among others, the commercial *PowerFlow* package is an example of a lattice Boltzmann (LB) solver being used in the production environment of car companies. The algorithm is interesting for GPU calculations due to its parallelizability. In [WZF<sup>+</sup>03], interactive wind calculations were performed on a GPU using the LBM, while in [WWXP06], interactive snow was simulated. Geist et. al even adapted the algorithm to light diffusion in [GRWS04]. Multi phase models for the LBM originally used smooth interface transitions, such as the immiscible model of Gunstensen et. al [GRZZ91]. This algorithm requires a recoloring of the participating fluids at the interface, and the broad interfaces require high resolutions to simulate small scale features. Several other methods exist for multi-phase flows, e.g., [SC94], [SOOY96] or [TKSR02, TFK03b]. These have also been extended in different ways, e.g., to allow for high density ratios [IBefitpfwldd04]. Ginzburg and Steiner, on the other hand, propose a VOF based free surface approach in [GS03] that is similar to the method that is used in the following. However, it requires additional computations to increase the accuracy of the free surface tracking. This thesis is based on the method that was applied and validated, e.g., in [KTH<sup>+</sup>05] to simulate metal foaming. A metal foam sample and a simulation result can be seen in Figure 2.1. The method was found to be both sufficiently accurate and computationally efficient. It will be extended in the following to allow the creation of flexible and realistic animations of free surface fluids.

## 2.2 Comparing Simulation Approaches

As for physically based animations a realistic appearance is the most important criterion, it is difficult to rigorously compare the aforementioned simulation approaches. For engineering applications the ability of an algorithm to accurately compute a certain flow property is usually used to choose an algorithm. This could be the accuracy of a drag and lift force computation around a given object. For physically based animations no such measurement values exist. In general, fluid simulations for physically based animation are required to handle low viscosities, thin sheets of fluid and small details such as drops or fine obstacles. Furthermore, the fluid movement should show no signs of compressibility or flickering. The algorithms should also exhibit high computational efficiency in order to be of practical use.

Figure 2.2 summarizes the advantages and disadvantages that were experienced during the work on this thesis with some of the typical fluid algorithms: the LB solver of this thesis, a typical semi-Lagrangian level set based NS solver, and a kd-tree based SPH solver. The main advantages of the level set solver are the smooth surface representation due to the level set, and the arbitrary time step size of the semi-Lagrangian

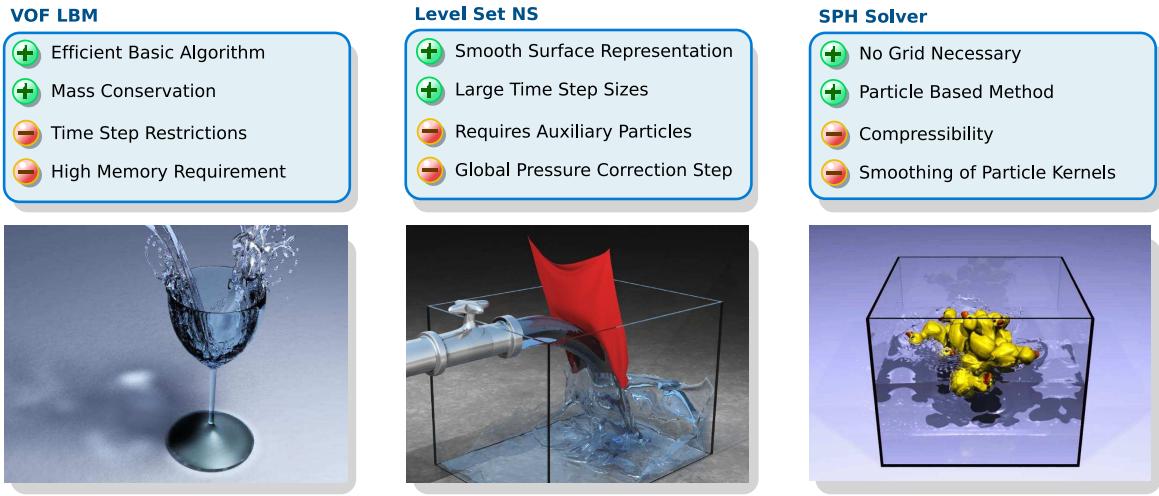


Figure 2.2: Here an overview of the properties of three different simulation approaches can be seen. Thanks to the authors of [KAG<sup>+</sup>06] and [GSLF05, Gue06] for the permission to use the SPH picture and level set picture, respectively.

solver. On the other hand, the level set tracking by itself is not mass conserving, and thus requires additional techniques to guarantee mass conservation. Furthermore, the pressure correction of an NS solver requires global information to achieve a divergence free velocity field. Even when the method from [Sta99] is used, the velocity field has to be globally corrected after each time step, while both the LBM and SPH explicitly solve the pressure during the course of the simulation. However, this leads to another problem with these two methods: if the time step size is too large, it can lead to a noticeable compressibility of the liquid. The main advantages of an SPH solver are its particle based representation of the fluid, and its natural ability to handle free surfaces in a mass conserving way. A drawback of SPH for detailed fluids is the increased computational requirement for large particle numbers, unless a technique such as the multi-resolution approach of [KAG<sup>+</sup>06] is used. Lastly, the LBM in combination with the VOF free surface method is mainly attractive due to its efficient basic algorithm with its local handling of boundary conditions. Due to the VOF method, it is also fully mass conserving. The main drawbacks that are encountered when using LB solvers, are the increased memory requirements and the small time step size in comparison to NS solvers. On the other hand, it should be noted that a single LB step is usually significantly faster than the update step of an NS solver.

In conclusion of this brief comparison, it can be stated that all previously mentioned simulation approaches have their drawbacks and advantages. In accordance to the *no-free-lunch* theorems, an efficient and flexible free surface simulator requires a combination of state of the art algorithms for each class of solvers. The overview given in the previous paragraph should only be taken as a general trend. In fact, even different implementations of the same algorithm can exhibit strongly varying behavior. During the work on this thesis it became clear that the LBM is a valid alternative to other simulation algorithms. Its interesting properties and the quickly growing amount of research in the area make it an approach that is worth considering for a wide variety

of applications.

## Outline

This thesis will first give a brief description of the basic LB algorithm. It will be explained how to derive the algorithm from the Boltzmann equation, and it will be shown that the LBM yields an approximation to the Navier-Stokes equations. In Chapter 4 the VOF model for the LBM will be explained. Additionally, chapter 5 will present algorithms to efficiently handle moving obstacles for free surface LB simulations. Next, an explanation will be given of how to adapt the time step size during the course of the simulation. Chapter 7 will introduce a combination of all previous algorithms with a method to adaptively coarsen the grid for large fluid volumes. As this algorithm represents the final state of the free surface LBM itself, a performance evaluation and comparison will be given at this point. Chapter 8 will then explain how to parallelize the method, and show performance results for the parallelized version. The next three chapters will focus on extensions that are targeted towards extending the flexibility of the algorithm. Chapter 9 concentrates on controlling fluid simulations, without disturbing important details of the flow. Chapter 10 discusses an extension to the free surface algorithm that enables simulations of large scale open water scenes. (This is done by coupling the three-dimensional simulation to a two-dimensional shallow water simulation in combination with a model to animate small drops and foam.) Then Chapter 11 will demonstrate how to construct a general API for the integration of a free surface solver into a 3D application. Finally, in Chapter 12 a summary and an outlook of possible extensions of the algorithm will be given.

As the topic of this thesis is the *animation* of fluids, the still pictures shown in the following cannot fully capture the visual appearance of the corresponding simulations. Hence, the animations of the test cases were made available on the internet under [www.ntoken.com/fluid](http://www.ntoken.com/fluid).



$$1.380650 * 10^{-23} \text{ [m}^2 \text{ kg / (s}^2 \text{ K}^1\text{)]}$$

*Boltzmann Constant*

# Chapter 3

## The Lattice Boltzmann Method

This chapter will describe the basic algorithm of the LBM. First, an overview of the historical development of the method will be given. Next, the method itself and the parametrization will be described. Finally, a derivation of the necessary equations and a turbulence model to ensure stability are explained.

### 3.1 Historical Development

The free surface simulations of this thesis are based on the LBM, which means that the simulation region is divided into a Cartesian (and in this case, equidistant) grid of cells, each of which only interacts with cells in its direct neighborhood. While conventional solvers directly discretize the NS equations, the LBM is essentially a first order explicit discretization of the Boltzmann equation in a discrete phase-space. It can also be shown, that the LBM approximates the NS equations with good accuracy, an overview of this derivation will be given later in this chapter. A detailed overview of the LBM can be found, e.g., in [WG00, Suc01].

The Boltzmann equation itself has been known since 1872. It is named after the Austrian scientist Ludwig Boltzmann, and is part of the classical statistical physics that describe the behavior of a gas on the microscopic scale. The LBM follows the approach of cellular automata to model even complex systems with a set of simple and local rules for each cell [Wol02]. As the LBM computes macroscopic behavior, such as the motion



of a fluid, with equations describing microscopic scales, it operates on a mesoscopic level in between those two extremes.

Historically, the LBM evolved from methods for the simulation of gases that computed the motion of each molecule in the gas purely with integer operations. In [HYP76], there was a first attempt to perform fluid simulations with this approach. It took ten years to discover that the isotropy of the lattice vectors is crucial for a correct approximation of the NS equations [FdH<sup>+</sup>87]. Motivated by this improvement, [MZ88] developed the first algorithm that was actually called LBM by performing simulations with averaged floating point values instead of single fluid molecules. The third important contribution to the basic LBM was the simplified collision operator with a single time relaxation parameter. This collision operator is known as the *Bhatnagar Gross Krook* (BGK) approximation [BGK54], and was derived independently by [CCM92], and [QdL92]. Since then, the LBM has been applied to many classes of fluid mechanics problems: the direct numerical simulation of turbulence [YGL05b], and Eulerian-Lagrangian simulations [MC98], among others. Moreover, the LBM is available in commercial fluid solvers [LLS00], which are in production use at, e.g., aerospace and car companies, to name some examples. As the LBM can handle problems with a wide range of Knudsen numbers, it is moreover interesting for problems where the NS equations are no longer applicable. It can, e.g., be applied to hypersonic or rarefied gas flows [SZ04].

A general comparison of LB solvers and conventional NS solvers is difficult. However, [GKT<sup>+</sup>06] compared state of the art solvers of both kinds. They come to the conclusion that there is no clear winner, but the computational efficiency of the LBM solver is comparable to a discretization of the corresponding NS problem. For special types of problems, each type of solver has its advantages and disadvantages. Overall, a simple LB implementation performs very well for complex geometries, as each LB cell contains information not only about the fluid velocity and pressure, but also about their spatial derivatives. The method thus allows an accurate representation of obstacles even for coarse grids. The free surface of a fluid usually results in complex and, in particular, time dependent topologies. This is the motivation to use LBM for simulating free surface flows, and the model used in the following is especially simple due to this ability of the LBM to model complex boundary conditions. It will be discussed in more detail in Chapter 4. The basic LBM algorithm without extensions will be described and derived in the following sections.

## 3.2 The Basic Algorithm

The basic LB algorithm consists of two steps, the stream-step, and the collide-step. These are usually combined with no-slip boundary conditions for the domain boundaries or obstacles. The simplicity of the algorithm is especially evident when implementing it, which, for the basic algorithm, requires roughly a single page of C-code. Using a LBM, the particle movement is restricted to a limited number of directions. Here, a three-dimensional model with 19 velocities (commonly denoted as *D3Q19*) will be used. Alternatives are models with 15 or 27 velocities. However, the latter one has no apparent advantages over the 19 velocity model, while the model with 15

---

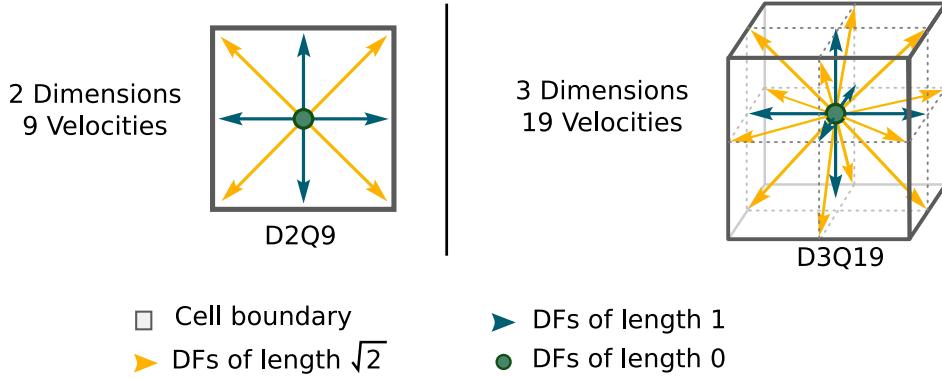


Figure 3.1: The most commonly used LB models in two and three dimensions.

velocities has a decreased stability. The D3Q19 model is thus usually preferable as it requires less memory than the 27 velocity model. For two dimensions the D2Q9 model with nine velocities is the most common one. The D3Q19 model with its lattice velocity vectors  $e_{1..19}$  is shown in Figure 3.1 (together with the D2Q9 model). The velocity vectors take the following values:  $e_1 = (0, 0, 0)^T$ ,  $e_{2,3} = (\pm 1, 0, 0)^T$ ,  $e_{4,5} = (0, \pm 1, 0)^T$ ,  $e_{6,7} = (0, 0, \pm 1)^T$ ,  $e_{8..11} = (\pm 1, \pm 1, 0)^T$ ,  $e_{12..15} = (0, \pm 1, \pm 1)^T$ , and  $e_{16..19} = (\pm 1, 0, \pm 1)^T$ . As all formulas for the LBM usually only depend on the so-called particle distribution functions (DFs), all of these two-dimensional and three-dimensional models can be used with the method presented here. To increase clarity, the following illustrations will all use the D2Q9 model.

For each of the velocities, a floating point number  $f_{1..19}$ , representing the fraction of particles moving with this velocity, needs to be stored. Thus, in the D3Q19 model there are particles not moving at all ( $f_1$ ), moving with speed 1 ( $f_{2..7}$ ) and moving with speed  $\sqrt{2}$  ( $f_{8..19}$ ). In the following, a subscript of  $\tilde{i}$  will denote the value from the inverse direction of a value with subscript  $i$ . Thus,  $f_i$  and  $f_{\tilde{i}}$  are opposite DFs with inverse velocity vectors  $e_{\tilde{i}} = -e_i$ . During the first part of the algorithm (the stream step), all DFs are advected with their respective velocities. This propagation results in a movement of the floating point values to the neighboring cells, as shown in Figure 3.2. Formulated in terms of DFs the stream step can be written as

$$f_i^*(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x} + \Delta t e_{\tilde{i}}, t). \quad (3.1)$$

Here,  $\Delta x$  denotes the size of a cell and  $\Delta t$  the time step size. Both are normalized by the condition  $\Delta t/\Delta x = 1$ , which makes it possible to handle the advection by a simple copying operation, as described above. These post-streaming DFs  $f_i^*$  have to be distinguished from the standard DFs  $f_i$ , and are never really stored in the grid. The stream step alone is clearly not enough to simulate the behavior of incompressible fluids, which is governed by the ongoing collisions of the particles with each other. The second part of the LBM, the collide step, amounts for this by weighting the DFs of a cell with the so called *equilibrium distribution functions*, denoted by  $f_i^{eq}$ . These depend solely on the density and velocity of the fluid. Here, the incompressible model from [HL97b] is used, which alleviates compressibility effects of the standard model by using a modified equilibrium DF and velocity calculation. The density and velocity can

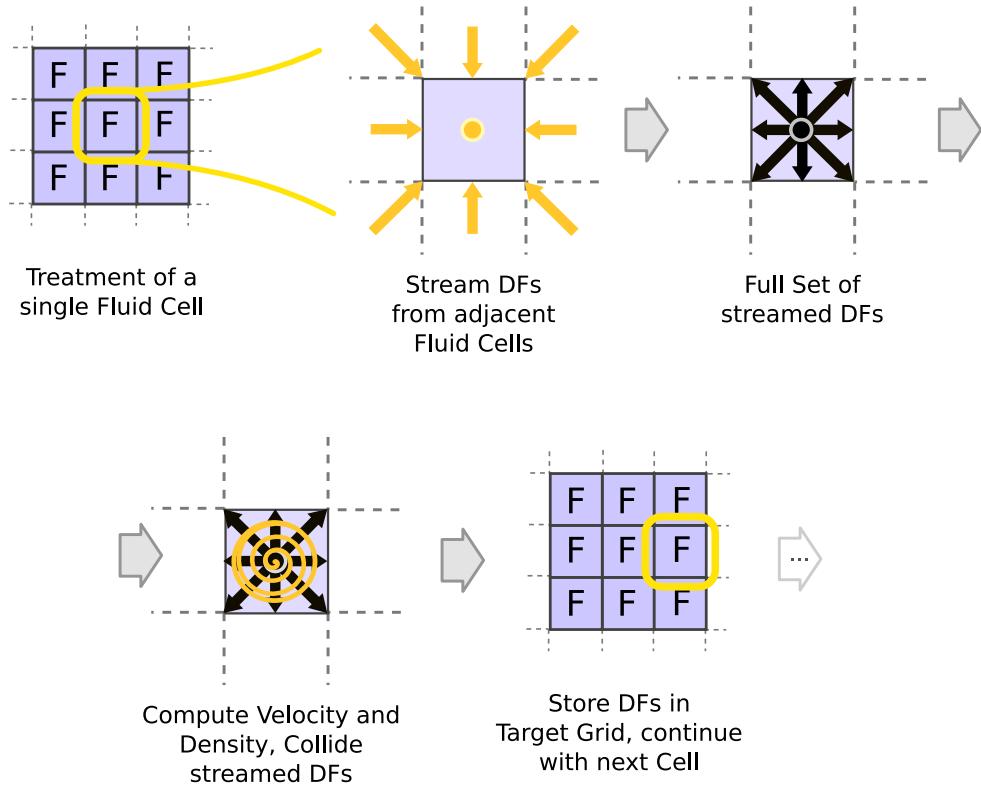


Figure 3.2: This figure gives an overview of the stream and collide steps for a fluid cell.

be computed by summation of all the DFs for one cell

$$\rho = \sum f_i \quad \mathbf{u} = \sum \mathbf{e}_i f_i . \quad (3.2)$$

The standard model, in contrast to the one used here, requires a normalization of the velocity with the fluid density. For a single direction  $i$ , the equilibrium DF  $f_i^{eq}$  can be computed with

$$\begin{aligned} f_i^{eq} &= w_i \left[ \rho + 3\mathbf{e}_i \cdot \mathbf{u} - \frac{3}{2}\mathbf{u}^2 + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{u})^2 \right], \quad \text{where} \\ w_i &= 1/3 \quad \text{for } i = 1, \\ w_i &= 1/18 \quad \text{for } i = 2..7, \\ w_i &= 1/36 \quad \text{for } i = 8..19. \end{aligned} \quad (3.3)$$

The equilibrium DFs represent a stationary state of the fluid. However, this does not mean that the fluid is not moving, but only that the values of the DFs would not change, if the whole fluid was at an equilibrium state. For very viscous flows, such an equilibrium state (equivalent to a *Stokes* flow) can be globally reached. In this case, the DFs will converge towards constant values. The collisions of the molecules in a real fluid are approximated by linearly relaxing the DFs of a cell towards their equilibrium state. Thus, each  $f_i$  is weighted with the corresponding  $f_i^{eq}$  using:

$$f_i(\mathbf{x}, t + \Delta t) = (1 - \omega)f_i^*(\mathbf{x}, t + \Delta t) + \omega f_i^{eq}. \quad (3.4)$$

Here,  $\omega$  is the parameter that controls the viscosity of the fluid. Often,  $\tau = 1/\omega$  is also used to denote the lattice viscosity. The parameter  $\omega$  is in the range of  $(0..2]$ , where values close to 0 result in very viscous fluids, while values near 2 result in more turbulent flows. Usually these are also visually more interesting. However, for values close to 2, the method can become unstable. In Section 3.3, a method to stabilize the computations with a turbulence model will be explained. This alleviates the instabilities mentioned above. The parameter  $\omega$  is given by the kinematic viscosity of a fluid. Details of the parametrization will be explained in Section 3.4. The values computed with Equation (3.4) are stored as DFs for time  $t + \Delta t$ . As each cell needs the DFs of the adjacent cells from the previous time step, two arrays for the DFs of the current and the last time step are usually used.

The easiest way to implement the no-slip boundary conditions is the link bounce back rule that results in a placement of the boundary halfway between fluid and obstacle cells. If the neighboring cell at  $(\mathbf{x} + \Delta t \mathbf{e}_i)$  is an obstacle cell during streaming, the DF from the inverse direction of the current cell is used. Thus, Equation (3.1) changes to

$$f_i^*(\mathbf{x}, t + \Delta t) = f_{\bar{i}}(\mathbf{x}, t). \quad (3.5)$$

Figure 3.3 illustrates those basic steps for a cell next to an obstacle cell.

An implementation of the algorithm described so far might consist of a flag field to distinguish fluid and obstacle cells, and two arrays of single-precision floating point variables, with 19 values for each cell in the grid. During a loop over all cells in the current grid, each cell collects the neighboring DFs according to Equation (3.1) or Equation (3.5), for adjacent fluid and obstacle cells respectively. The density and velocity are computed and used to calculate the equilibrium DFs. These are weighted with the streamed DFs and written into the other grid, continuing with the next cell in the grid. Subsequent time steps alternate in streaming and colliding the DFs from one grid array to the other. Note that using Equation (3.5) the DFs for obstacle cells are never touched.

In contrast to a standard finite-difference NS solver, the implementation is much simpler, but also requires more memory. Such a typical NS solver usually requires 7 floating point values for each grid point (pressure, three velocity components, plus three temporary variables), but for some cases it might need higher resolutions to resolve obstacles with the same accuracy. Using a more sophisticated LB implementation with *grid compression* [PKW<sup>+</sup>03], the memory requirements can be reduced to almost half of the usual requirements. Furthermore, using an adaptive time step size is common practice for a NS solver, while the size of the time step in the LBM is, by default, fixed to 1 (although in Chapter 6 a method to also change the time step size for a LB solver will be presented). As the maximum lattice velocity may not exceed  $1/3$ , in order for the LBM to remain stable, it might still need several time steps to advance to the same time a NS solver would reach in a single step. However, each of these time steps usually requires a significantly smaller amount of work, as the LBM can be computed very efficiently on modern CPUs. Moreover, it does not require additional global computations such as the pressure correction step.

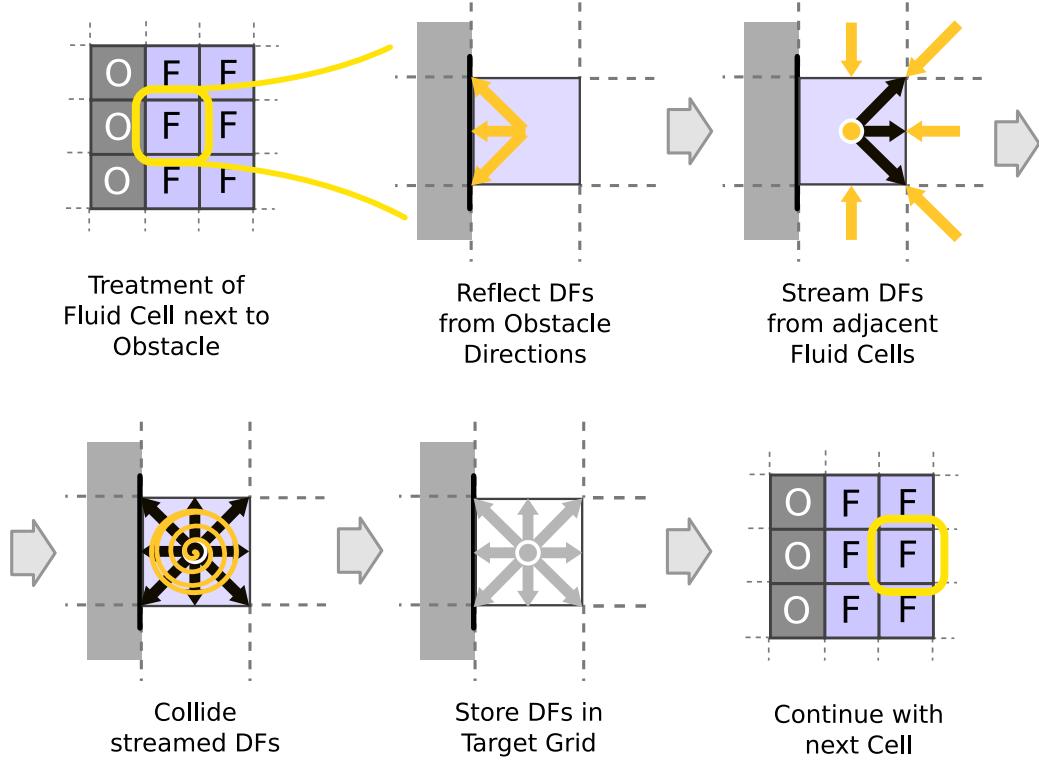


Figure 3.3: This figure gives an overview of the stream and collide steps for a fluid cell next to an obstacle.

### 3.3 Stability

In order to simulate turbulent flows with the LBM, the basic algorithm needs to be extended, as its stability is limited once the relaxation parameter  $\tau$  approaches  $1/2$  (which is equivalent to  $\omega$  being close to 2). Here, the Smagorinsky sub-grid model, as used in, e.g., [WZF<sup>+</sup>03, LWK03], will be applied. Its primary use is to stabilize the simulation, instead of relying on its ability to accurately model subgrid scale vortices in the simulation. Compared to the small slowdown due to the increased complexity of the collision operator, this usually results in a large improvement of efficiency, as the simulations would otherwise require considerably finer grid resolutions.

The sub-grid turbulence model applies the calculation of the local stress tensor as described in [Sma63] to the LBM. The computation of this tensor is relatively easy for the LBM, as each cell already contains information about the derivatives of the hydrodynamic variables in each DF. The magnitude of the stress tensor is then used in each cell to modify the relaxation time according to the eddy viscosity. For the calculation of the modified relaxation time, the Smagorinsky constant  $C$  is used. For the simulations in the following,  $C$  will be set to 0.03. Values in this range are commonly used for LB simulations, and were shown to yield good modeling of the sub-grid vortices [YGL05a]. The turbulence model is integrated into the basic algorithm that was described in Section 3.2 by adding the calculation of the modified relaxation time after the streaming step, and using this value in the normal collision step.

The modified relaxation time  $\tau_s$  is calculated by performing the steps that are described in the following. First, the non-equilibrium stress tensor  $\Pi_{\alpha,\beta}$  is calculated for

each cell with

$$\Pi_{\alpha,\beta} = \sum_{i=1}^{19} \mathbf{e}_{i_\alpha} \mathbf{e}_{i_\beta} (f_i - f_i^{eq}), \quad (3.6)$$

using the notation from [HSCD96]. Thus,  $\alpha$  and  $\beta$  each run over the three spatial dimensions, while  $i$  is the index of the respective velocity vector and DF for the D3Q19 model. The intensity of the local stress tensor  $\mathcal{S}$  is then computed as

$$\mathcal{S} = \frac{1}{6C^2} \left( \sqrt{\nu^2 + 18C^2 \sqrt{\Pi_{\alpha,\beta} \Pi_{\alpha,\beta}}} - \nu \right). \quad (3.7)$$

Now the modified relaxation time is given by

$$\tau_s = 3(\nu + C^2 \mathcal{S}) + \frac{1}{2}. \quad (3.8)$$

From Equation (3.7) it can be seen that  $\mathcal{S}$  will always have a positive value – thus the local viscosity will be increased depending on the size of the stress tensor calculated from the non-equilibrium parts of the distribution functions of the cell to be relaxed. This effectively removes instabilities due to small values of  $\tau$ . Note that for engineering applications it is usually important to evaluate the effects of the turbulence model on accuracy, while it is primarily applied for stability in this thesis.

A fundamentally different method that also leads to a stabilization of the basic LBM is the multi relaxation time approach (MRT). In this case, the single time relaxation operator of Equation (3.4) is replaced by a more advanced one that relaxes the different hydrodynamic moments separately [LL00, dGK<sup>+</sup>02]. These multiple relaxation times can be used to increase the accuracy of the simulation, e.g., for precisely handled obstacles in the flow. Moreover, it can be combined with the Smagorinsky turbulence model, as demonstrated by Krafczyk et. al in [KTL03] or by Yu et. al in [YLG06]. However, for the rest of this thesis, the single relaxation time model will be used, as the increased accuracy of MRT is not necessary, and the additional computations would lead to a slightly reduced performance.

## 3.4 Parametrization

Below, the conversion of dimensional quantities, denoted by primed symbols, into dimensionless quantities used in the LBM will be described. Given the real-world values for kinematic viscosity  $\nu' [\frac{m^2}{s}]$ , domain size  $S' [m]$ , a desired grid resolution  $r$ , and a gravitational force  $\mathbf{g}' [\frac{m}{s^2}]$ , the corresponding lattice quantities are computed as described in the following. Let  $S'$  be the length of one side of the domain that should be resolved with  $r$  cells. The cell size used by the LBM can then be computed as  $\Delta x' = S'/r$ .

The dimensional timestep  $\Delta t'$  is computed by limiting the compressibility due to gravitational force. Here a value of  $g_c = 0.005$  is used to keep the compressibility below half a percent. Thus,

$$\Delta t' = \sqrt{\frac{g_c \cdot \Delta x'}{|\mathbf{g}'|}} \quad (3.9)$$

yields a time step ensuring that the force exerted on each cell due to gravitational acceleration is causing less than a factor  $g_c$  of compression. Given  $\Delta x'$  and  $\Delta t'$ , the dimensionless lattice viscosity  $\nu$ , and relaxation time  $\omega$  are computed as

$$\nu = \nu' \frac{\Delta t'}{\Delta x'^2}, \quad (3.10)$$

and

$$\tau = 3\nu + 1/2, \quad \omega = \frac{1}{\tau}. \quad (3.11)$$

Likewise, the lattice acceleration  $\mathbf{g}$  (e.g., due to gravity) is calculated as

$$\mathbf{g} = \mathbf{g}' \frac{\Delta t'^2}{\Delta x'}. \quad (3.12)$$

In conclusion, a valid parametrization for an LB fluid simulation is given by the physical scale, the desired kinematic viscosity and the compressibility factor. For a given grid resolution, the values of  $\tau$  and  $\Delta t$  can then be calculated. Note that when the grid resolution is small in combination with a low viscosity, the resulting value of  $\tau$  will be close to 1/2. Due to the turbulence model of Section 3.3, the simulation will remain stable, but effectively increase the viscosity to a value that can be handled with the chosen grid resolution.

## 3.5 Derivation

This section will give an overview of the Navier-Stokes and the Boltzmann equations and relate them to the LBM. Furthermore, the a priori derivation of the LBM will be described.

### 3.5.1 The Navier-Stokes Equations

The origins of the established Navier-Stokes (NS) equations reach back to Isaac Newton, who, around 1700, formulated the basic equations for the theoretical description of fluids. These were used by L. Euler half a century later to develop the basic equations for momentum conservation and pressure. Amongst others, Louis M. H. Navier continued to work on the fluid mechanic equations at the end of the 18th century, as did Georg G. Stokes several years later. He was one of the first to analytically solve fluid problems for viscous media. The NS equations could not be practically used until in the middle of the 20th century, when the numerical methods, that are necessary to solve the resulting equations, were developed. Further information on fluid mechanics in general and the NS equations can for example be found in [KC04].

One of the important aspects for a fluid mechanics problem is the conservation of mass, as the mass of the fluid naturally has to remain constant for a given fluid system. This is ensured by the continuity equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0. \quad (3.13)$$

Note that the second partial derivative is written in Einstein notation, meaning that the  $i$  subscript appearing twice denotes a sum over all possible values. In this case it is a sum over the three spatial dimensions. For a fluid with constant density, Equation (3.13) can be simplified to

$$\frac{\partial u_i}{\partial x_i} = 0, \quad (3.14)$$

which means that the velocity field has to be divergence free in order to conserve mass.

The continuity equation is applied together with the NS equations, or momentum equations, which can be written as:

$$\underbrace{\rho \left( \frac{\partial u_j}{\partial t} + u_i \frac{\partial u_j}{\partial x_i} \right)}_{\text{advection}} + \underbrace{\frac{\partial P}{\partial x_j}}_{\text{pressure}} + \underbrace{\frac{\partial \tau_{ij}}{\partial x_i}}_{\text{momentum}} = \rho g_j, \quad j = 1, 2, 3. \quad (3.15)$$

Three parts of this equation can be distinguished. The first part of the equation is responsible for mass forces like advection. The partial derivatives of the pressure  $P$  are surface forces acting on the fluid. The third, and most complicated part, contains the tensor  $\tau_{ij}$ , and introduces momentum effects due to molecular movement. This effect is similar to a friction between the fluid layers, but can be attributed to the momentum exchange due to Brownian motion of the molecules [Dur06].

For Newtonian fluids, i.e., fluids with a viscosity that is independent of the shear rate,  $\tau_{ij}$  can be computed as follows

$$\tau_{ij} = -\mu \left( \frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right) + \frac{2}{3} \delta_{ij} \mu \frac{\partial u_k}{\partial x_k}. \quad (3.16)$$

In this equation,  $\mu$  denotes the dynamic shear viscosity, a value depending on the physical properties of the fluid. Note that  $\nu$  denotes the kinematic viscosity, which is related to the dynamic viscosity by  $\nu = \mu/\rho$ . The Kronecker symbol denotes a tensor with  $\delta_{ij} = 1$  for  $i = j$ , and  $\delta_{ij} = 0$  otherwise. As  $\tau_{ij}$  can be computed with Equation (3.16), this leaves six unknown variables in Equation (3.14) and Equation (3.15): the pressure, the three velocity components, the density and the viscosity. However, for incompressible ( $\rho = \text{const}$ ) and Newtonian ( $\mu = \text{const}$ ) fluids the two remaining unknowns pressure and velocity can be solved with the simplified equations:

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (\text{continuity equation}) \quad (3.17)$$

$$\rho \left( \frac{\partial u_j}{\partial t} + u_i \frac{\partial u_j}{\partial x_i} \right) + \frac{\partial P}{\partial x_j} = \mu \frac{\partial^2 u_j}{\partial x_i^2} + \rho g_i \quad (\text{NS equations}). \quad (3.18)$$

With adequate initial and boundary conditions, these equations can be discretized, e.g., using finite differences or finite volumes, and solved using numerical algorithms such as Gauss-Seidel, conjugate gradient or multigrid methods. An implementation of a finite-difference discretization of the NS equations with explicit time stepping can be found, e.g., in [GDN88].

In fluid mechanics, these equations are usually treated in a dimensionless way. This is valid as fluids behave similar at different size and time scales when they have the

same *Reynolds number* ( $Re$ ). It is a dimensionless value, and can be calculated in the following way:

$$Re = \frac{U L}{\nu} \quad (3.19)$$

Here  $\rho$  is the fluid density,  $U$  the macroscopic flow speed and  $L$  the characteristic length or distance of the problem. Thus, a fluid with a given velocity and viscosity behaves similar to one with a lower velocity and correspondingly smaller viscosity. Similarly, two problems with the same physical fluids are comparable when the flow speed is increased, and the characteristic length is decreased by the same factor. For example, in order to measure the flow around an aerodynamic body, wind tunnels with a smaller model and increased flow speed are often used. More details on the NS equations, their derivation and applications can be found in many textbooks on fluid dynamics, such as [Dur06] and [KC04].

### 3.5.2 The Boltzmann Equation

Including an external force  $G$ , the Boltzmann equation can be written as

$$\frac{\partial f}{\partial t} + \xi \cdot \frac{\partial f}{\partial \mathbf{x}} + \mathbf{G} \cdot \frac{\partial f}{\partial \xi} = \Omega(f) , \quad (3.20)$$

where the function  $f$  gives the amount of particles traveling with a given speed, volume, time and position. The left hand side describes the overall motion of the molecules with the microscopic velocity  $\xi$  through the force field that is given by  $G$  at  $x$ , while the right hand side models the interaction of molecules with the collision operator  $\Omega$ . It is an integral equation that includes the differential collision cross section for the two particles, which can be calculated geometrically by approximating the molecules with rigid spheres for the collision [Fro79].

Due to the complicated nature of the collision operator  $\Omega$ , it is often replaced by simpler expressions that preserve the collision invariants and tend towards a Maxwellian distribution. The standard model for this is the BGK approximation that was proposed in [BGK54] and [Wel54]:

$$\Omega_{BGK}(f) = \frac{f^e - f}{\tau} \quad (3.21)$$

Here  $f^e$  is a Maxwellian distribution representing the *local equilibrium* that is parametrized by the conserved quantities density  $\rho$ , speed  $\xi$  and temperature  $T$ . Each collision changes the distribution function  $f$  proportional to the departure from the local equilibrium  $f^e$ , where the amount of this correction is modified by the relaxation time  $\tau$ . In general, the collision time is dependent on properties of the gas and its current state. However, for the BGK approximation, it is simplified and represented as a single value.

The local equilibrium is reached when  $\Omega(f^e, f^e)$  vanishes. With this property, it can be shown that  $f$  is collision invariant, and, as such, does not change under the effects of a collision. The density  $\rho$ , momentum  $\xi_a$ , and energy  $E$  are the Lagrangian parameters. Assuming a normalized particle mass of 1, they are computed as

$$\int f d\xi = \rho , \quad \int f u_a d\xi = \rho \xi_a , \quad \text{and} \quad \int f \frac{u^2}{2} d\xi = \rho E . \quad (3.22)$$

The macroscopic flow speed  $\xi_a$ , density  $\rho$ , and fluid temperature  $T$  parametrize the Maxwell distribution (sometimes also called Maxwell-Boltzmann distribution). For three dimensions it is:

$$f^M = \rho \left( \frac{m^2}{2\pi RT} \right)^{3/2} e^{-\frac{(\xi - \mathbf{u})^2 m^2}{2RT}} \quad (3.23)$$

Where  $R$  is the Boltzmann constant, and  $m$  the mass of a particle.

### 3.5.3 Chapman-Enskog Expansion

To show that the Boltzmann equation can be used to describe fluids, the NS equations are derived by a multi-scale analysis called Chapman-Enskog expansion. It relies on the Knudsen number  $Kn = \lambda/L_C$ , which is the ratio between the mean free path length  $\lambda$  and the characteristic shortest scale of the macroscopic system that needs to be considered ( $L_C$ ). The Knudsen number has to be much smaller than one, in order for the treatment of the fluid as a continuous system to be valid. For the derivation of the NS equations from the Boltzmann equation, the latter is split according to a hierarchy of different scales for space and time variables. It is based on the expansion parameter  $\varepsilon$  for which the Knudsen number  $Kn$  will be used. Usually, the expansion is truncated after terms of second order. In the following, a derivation of the Euler equations will be shown, which also illustrates the subsequent steps that are necessary to derive the full NS equations.

For the time variables, the representation  $t = \varepsilon t_1 + \varepsilon^2 t_2$  is chosen. The time  $t$  represents the fast local relaxations in a fluid by collisions. Sound waves, as well as advection, are of the scale  $t_1$ , and are considerably slower than the local relaxations. Still, these are faster than diffusion processes which take place on the time scale  $t_2$ . On the other hand, only one spatial expansion has to be considered, giving the following expansion of first order:  $\mathbf{x} = \varepsilon \mathbf{x}_1$ . This is due to the fact that advection and diffusion are both considered in similar spatial scales  $x_1$ . The differential operators are represented in the same way

$$\frac{\partial}{\partial x_a} = \varepsilon \frac{\partial}{\partial x_a}, \quad \text{and} \quad \frac{\partial}{\partial t} = \varepsilon \frac{\partial}{\partial t} + \varepsilon^2 \frac{\partial}{\partial t}. \quad (3.24)$$

For a consistent expansion, the second order terms in space are also necessary. The moment equations of  $f$  are directly expanded to a sum of the form:

$$f = \sum_{n=0}^{\infty} \varepsilon^n f^n \quad (3.25)$$

Furthermore, it is assumed that the time dependence of  $f$  is only caused by the variables  $\rho$ ,  $\mathbf{u}$ , and  $T$ . Expanding Equation (3.20) in both space and time up to second order yields

$$\varepsilon \frac{\partial f}{\partial t_1} + \varepsilon^2 \frac{\partial f}{\partial t_2} + \varepsilon u_a \cdot \frac{\partial f}{\partial x_a} + \frac{1}{2} \varepsilon^2 u_a u_b \frac{\partial^2 f}{\partial x_a \partial x_b} = \Omega(f^0) + \varepsilon \frac{\partial \Omega(f^1)}{\partial f}. \quad (3.26)$$

Note that here  $f^0$  is a Maxwell distribution, and as such, due to the definition of the BGK collision approximation in Equation (3.21),  $\Omega(f^0)$  is zero. The three scales from

$\mathcal{O}(\varepsilon^0)$  to  $\mathcal{O}(\varepsilon^2)$  can be distinguished in Equation (3.26), and are handled separately. In the following, subsequent expansions of the conservation equations will be performed. Using a second order Knudsen number expansion of the mass  $m$ , the first order terms of Equation (3.26) yield

$$\frac{\partial \rho}{\partial t_1} + \frac{\rho \partial u_a}{\partial x_{1a}} = 0, \text{ and} \quad (3.27)$$

$$\frac{\partial \rho u_a}{\partial t_1} + \frac{\partial \int u_a u_b f^0 d\mathbf{u}}{\partial x_{1b}} = 0. \quad (3.28)$$

The continuity equation is already recognizable. When the integral of the second equation is evaluated analytically, it can be replaced by  $\rho u_a u_b + \rho T \delta_{ab}$  which leads to

$$\frac{\partial \rho u_a}{\partial t_1} + \frac{\partial \rho u_a u_b}{\partial x_{1b}} + \frac{\partial \rho T \delta_{ab}}{\partial x_{1b}} = 0, \quad (3.29)$$

yielding the Euler equation for inviscid flows without dissipation.

Finally, to derive the NS equations, the second order equations also have to be considered. For these, both equilibrium, and non-equilibrium levels have to be handled in the expansion. Still, using first order conservation terms of zero, and restoring the continuous form of the equations, the NS equations as in Equation (3.15), can be derived. This is possible as terms of order  $\mathcal{O}(u^3)$  can be neglected, due to the assumption of small Mach numbers for the expansion. The full derivation with these additional steps that are similar to the expansion steps shown above, is given in, e.g., [Har71, WG00].

### 3.5.4 Derivation of the Lattice Boltzmann Equation

The following section will explain the derivation of the lattice Boltzmann equation from the continuous Boltzmann equation. It is based on [HL97a], and the more detailed description in [Tre02]. The method described here allows the derivation of the Lattice Boltzmann equation from an arbitrary kinetic equation, although it historically derived from the Lattice Gas cellular automata. The following abbreviations will be used from in this chapter:  $f(\mathbf{x}, \xi, t) = f(t)$ , and  $f(\mathbf{x} + \xi a, \xi, t+a) = f(t+a)$ . The same abbreviations hold for  $g$ , which denotes an equilibrium distribution function. This is explained in more detail in Section 3.5.4.

As a starting point, the Boltzmann equation with BGK collision approximation will be used:

$$\frac{\partial f(t)}{\partial t} + \xi \frac{\partial f(t)}{\partial \mathbf{x}} = -\frac{1}{\lambda} [f(t) - g(t)] \quad (3.30)$$

where  $f$  is the particle distribution function at time  $t$  and position  $\mathbf{x}$  for the microscopic velocity  $\xi$ .  $1/\lambda = A \cdot n$  is the relaxation time for the collision that is calculated from the number of particles  $n$  and the proportional coefficient  $A$ . Here, the collision term has been linearized according to Equation (3.21) for simplicity, without loosing generality.  $g$  is the Maxwell distribution  $f^M$  from Equation (3.23).

The hydrodynamic properties of the fluid, the density  $\rho$ , velocity  $\mathbf{u}$ , and the temperature  $T$  can be calculated with the *moments* of the function  $f$ . Here, the energy  $\gamma$

from the energy density  $\rho\gamma$  can be used to determine the temperature of the fluid.

$$\rho = \int f(\mathbf{x}, \xi, t) d\xi \quad (3.31)$$

$$\rho\mathbf{u} = \int \xi f(\mathbf{x}, \xi, t) d\xi \quad (3.32)$$

$$\rho\gamma = \int \frac{1}{2}(\xi - \mathbf{u})f(\mathbf{x}, \xi, t) d\xi \quad (3.33)$$

Note that the equilibrium distribution function  $g$  is calculated with these hydrodynamic moments, although it is written as a function of time and velocity. Hence, these values have to be correctly approximated after discretization.

### Time discretization

Equation (3.30) can be formulated as an ordinary differential equation (ODE):

$$\frac{Df}{Dt} + \frac{1}{\lambda}f = \frac{1}{\lambda}g, \text{ where } \frac{D}{Dt} = \frac{\partial}{\partial t} + \xi \frac{\partial}{\partial \mathbf{x}} \quad (3.34)$$

is the time derivative along the microscopic velocity. Assuming that  $\delta_t$  is small and  $g$  is a smooth function, Equation (3.34) can be simplified to:

$$f(t + \delta_t) - f(t) = -\frac{\delta_t}{\lambda} (f(t) - g(t)). \quad (3.35)$$

Here, the relaxation time  $\frac{\delta_t}{\lambda}$  is usually written as  $\frac{1}{\tau}$ . This formula is already similar to Equation (3.4) above. The following sections will describe the discretization of the velocity space and the derivation of an equilibrium function  $g$  that is consistent with the Navier-Stokes equations.

### Approximation of the equilibrium distribution

The Maxwell distribution that is used as the equilibrium distribution function  $g$  is given by Equation (3.23). For a particle mass of 1 and  $D$  dimensions it reads:

$$g(\mathbf{u}) = \frac{\rho}{(2\pi RT)^{D/2}} e^{-\frac{(\xi-\mathbf{u})^2}{2RT}} \quad (3.36)$$

This function will be Taylor expanded in  $\mathbf{u}$  up to the second order, which is a valid approximation for small velocities, and low Mach numbers. The following formula will be used as the local equilibrium distribution for the following derivations:

$$f^{(eq)} = \frac{\rho}{(2\pi RT)^{D/2}} e^{-\frac{\xi^2}{2RT}} \left( 1 + \frac{\xi \cdot \mathbf{u}}{RT} + \frac{(\xi \cdot \mathbf{u})^2}{2(RT)^2} - \frac{\mathbf{u}^2}{2RT} \right). \quad (3.37)$$

It is derived by expanding the quadratic form in the exponent of  $e$  from Equation (3.36) and Taylor expanding the resulting equation.

## Discretization of the velocities

For simplicity, the D2Q9 model will be derived in the following section. As can be seen in Equation (3.31), the moment integrals over the whole velocity space have to be evaluated. As the velocity is not yet discretized, these run from  $-\infty$  to  $+\infty$  in both x and y direction for a two-dimensional model. The moments of the particle distribution functions are important for consistency with the Navier-Stokes equations. Another important property that has to be retained by the discretization is the isotropy, which is the most important of the Navier-Stokes symmetries. Therefore the lattice should be invariant to rotations of the problem – this can be shown by isotropy tensors as in [WG00]. For the LB derivation, the moments are directly used as constraint for the numerical integration method.

For models that include temperature the integration of moments up to second order has to be included. As an isothermal model will be used here, only the first moment, the velocity is required. The moments of Equation (3.37) in two dimensions can generally be written as follows:

$$I = \int \psi(\xi) f^{(0)} d\xi \quad \text{with} \quad \psi(\xi) = \xi_x^m \xi_y^n \quad (3.38)$$

Here  $\psi$  is the moment function that contains powers of the velocity components. After restructuring the equation, moments of up to the third order will occur in the equation - one from the velocity moment, and two from the  $(\xi \cdot \mathbf{u})^2$  term. For numerical treatment, Equation (3.38) can be written as

$$I = \frac{\rho}{(2\pi RT)^{D/2}} \int \psi(\xi) e^{-\frac{\xi^2}{2RT}} \left( 1 + \frac{\xi \cdot \mathbf{u}}{RT} + \frac{(\xi \cdot \mathbf{u})^2}{2(RT)^2} - \frac{\mathbf{u}^2}{2RT} \right) d\xi. \quad (3.39)$$

The next step for the derivation of the LBM is to numerically integrate these moments with

$$\int f(x) W(x) dx = \sum_{j=1}^N w_j f(x_j). \quad (3.40)$$

Here  $W(x)$  is the weighting function ( $e^{-x^2}$ , in this case), and  $f(x)$  is a polynomial in  $x$ , e.g.,  $f(\zeta_x) = \zeta_x^m$ . To numerically integrate functions such as  $e^{-\zeta^2}$ , the commonly used Gauss-Hermite quadrature can be applied [Bro05], which is correct for polynomials in  $W$  up to the order  $(2N - 1)$ . The order of the quadrature thus has to be chosen according to the order of the moment polynomial  $\psi$ . Although the model is isothermal, the energy due to the temperature has to be kept constant. Hence, there is no additional level of freedom for the temperature, but for the moment integration it has to be taken into account. A Gauss-Hermite quadrature of third order ( $N = 3$ ) is thus required

$$I_i^m = \sum_{j=1}^3 w_j (\zeta_j)^m. \quad (3.41)$$

The values of  $\zeta$  and  $w$  are given by the Gauss-Hermite quadrature as:  $\zeta_1 = -\sqrt{3/2}$ ,  $\zeta_2 = 0$ ,  $\zeta_3 = +\sqrt{3/2}$ ,  $w_1 = \sqrt{\pi}/6$ ,  $w_2 = 2\sqrt{\pi}/3$ , and  $w_3 = \sqrt{\pi}/6$ . The moment function

can again be shortened to:

$$I = \frac{\rho}{\pi} \sum_{i=1}^3 \sum_{j=1}^3 w_i w_j \psi(\zeta_{i,j}) \left( 1 + \frac{\xi \cdot \mathbf{u}}{RT} + \frac{(\xi \cdot \mathbf{u})^2}{2(RT)^2} - \frac{\mathbf{u}^2}{2RT} \right) \quad (3.42)$$

where  $\zeta_{i,j}$  is the vector given by the quadrature as  $\zeta_{i,j} = \sqrt{2RT} (\zeta_i, \zeta_j)^T$ . As the two sums run over three values for  $i$  and  $j$  each, there are a total of nine possible values for  $\zeta_{i,j}$  and  $w_i w_j$ . For these, a new single index will be introduced. Furthermore, a number of substitutions can be made. As an isothermal model is used, the temperature  $T$  can be replaced by a constant  $c = \sqrt{2RT} \sqrt{3/2} = \sqrt{3RT}$ . The speed of sound  $c_s = 1/\sqrt{3}$  in the model yields  $c_s^2 = c^2/3 = RT$ . The weights  $w$ , divided by  $\pi$  read:

$$\begin{aligned} w_0 &= w_2 w_2 = 4/9 \\ w_{1..4} &= w_1 w_2, w_2 w_1, w_3 w_2, w_2 w_3 = 1/9 \\ w_{5..8} &= w_1 w_3, w_3 w_1, w_1 w_1, w_3 w_3 = 1/36 \end{aligned} \quad (3.43)$$

Each component of the vectors  $\zeta_{i,j}$  is either 0 or  $\pm\sqrt{2RT}\sqrt{3/2} = \pm\sqrt{3RT} = c$ :

$$\begin{aligned} \mathbf{e}_0 &= \zeta_{1,1} = (0, 0)^T \\ \mathbf{e}_{1..4} &= \zeta_{1,2}, \zeta_{2,1}, \zeta_{3,2}, \zeta_{2,3}, = (\pm 1, 0)^T c, (0, \pm 1)^T c \\ \mathbf{e}_{5..8} &= \zeta_{1,3}, \zeta_{3,1}, \zeta_{1,1}, \zeta_{3,3}, = (\pm 1, \pm 1)^T c \end{aligned} \quad (3.44)$$

With these discrete velocities, Equation (3.42) reads:

$$I = \sum_{\alpha=1}^9 W_\alpha \psi(\mathbf{e}_\alpha) f_\alpha^{eq} \quad (3.45)$$

Here,  $W_\alpha$  can be identified as  $2\pi R T e^{\frac{\xi^2}{2RT}}$ . This yields the equilibrium distribution function already used as Equation (3.3) for each of the nine velocities:

$$f_\alpha^{eq} = w_\alpha \rho \left( 1 + \frac{3\mathbf{e} \cdot \mathbf{u}}{c^2} + \frac{9(\mathbf{e} \cdot \mathbf{u})^2}{2c^4} - \frac{3\mathbf{u}^2}{2c^2} \right) \quad (3.46)$$

Note that the lattice velocity vectors were given by the chosen Gauss-Hermite quadrature. The configuration of the lattice is likewise obtained from these velocities. It is possible to discretize velocities and lattice configuration differently, as has been shown in [HL97a], and [BdLL01].

Other LB models like the D3Q27 model can be derived in the same way. For the more often used three-dimensional model D3Q19, however, it is not possible to directly apply this method. Problems arise from the more irregular arrangement of the velocity vectors that cannot be easily formulated as a quadrature term. For these models, the *ansatz method* has to be used [WG00]. For a given kinetic equation like Equation (3.30) together with an equilibrium distribution, as the one from Equation (3.37), the velocity weights for a specific lattice can be calculated. Multi-scale analysis yields constraints for the moments of  $f$  that can be used to compute the required coefficients.

### 3.6 Closure

The LB algorithm developed from discrete gas simulations, and yields a simple yet efficient method to solve the NS equations on a mesoscopic scale. Parametrization of the lattice parameters can be performed by introducing a limit for the compressibility per time step. Furthermore, the Smagorinsky turbulence model is useful for stabilization of the algorithm, e.g., when fluids with a low viscosity (such as water) need to be simulated. With Chapman-Enskog expansion the NS equations can be derived from the LBM. In addition, an a priori derivation of the LBM from a given kinetic equation is possible by carefully discretizing velocity and time in combination with an appropriate equilibrium distribution function. The following chapters will extend the basic LBM described so far to perform free surface simulations.

A complex, abstract visualization of a fluid simulation. The scene is filled with swirling, translucent blue and white shapes representing liquid or gas. In the upper left, there is a block of text.

As he teetered on the Edge,  
the Waves were calling him.

Isis, The Beginning And The End

## Chapter 4

# Lattice Boltzmann Simulations with a Free Surface

Roughly three quarters of the earth are covered by water, and water is crucial for the survival of almost all lifeforms. As humans need to breathe air, we usually only see the interface between a volume of water, and the air, which itself can be regarded as a fluid. Water, and fluids in general, thus play an important role in everyday life, and are therefore important for any virtual application that is aiming to recreate a natural environment – be it a computer generated scene for a movie or a realtime computer game. This chapter will provide an algorithm to simulate a fluid with low viscosity, such as water, moving within a more viscous gas (e.g. air) – in terms of fluid mechanics, the water represents a fluid with a free surface. The free surface model of this thesis is similar to VOF approaches and thus explicitly conserves mass up to machine precision. It moreover includes the tracking of the fluid surface, hence no additional advection computations are necessary. The model itself, the boundary conditions and rules for cell conversion will be explained in this chapter. Afterwards results from interactive free surface simulations will be presented.

VOF methods are used for NS solvers since 1981 when they were introduced by Hirt and Nichols [HN81]. They can be used to achieve results of high quality, e.g., as shown in [Sus03] or [MMS04]. An overview of free surface simulations, and in more detail VOF, for engineering applications is given in [SZ99]. In contrast to the standard VOF methods the algorithm presented here directly computes the mass changes from the

---

values available in the LBM. Other multiphase and free surface LB models exist, such as [GRZZ91] and [GS03], but the boundary conditions presented in the following are especially inexpensive to compute. Thus, the algorithm achieves a high performance on common PC architectures. The regularity of the cell array results in a high cache efficiency, thus overcoming the memory bottleneck that often limits the performance [PKW<sup>03</sup>]. The algorithm furthermore performs very well in parallelized versions, as was shown, e.g., in [PTD<sup>04</sup>].

Most realtime simulations were up to now targeted towards the simulation of a single phase, e.g. to handle smoke. The semi-Lagrangian method was demonstrated for realtime use in [Sta03]. The model reduction technique of [TLP06] also allows realtime simulations. Moreover, it has the property to have a computational cost proportional to the number of points where the velocity field is evaluated. This is useful when only a limited number of particles should be traced in the velocity field of the fluid. Realtime fluid simulations without a free surface using the LBM were demonstrated, e.g., in [LWK03, WWXP06]. A full realtime free surface simulation method with the particle based SPH method was demonstrated in [MCG03a]. The algorithm presented in this chapter is based on a method originally developed to optimize and enhance the production process of metal foams [KS99, KS00, KBA<sup>00</sup>, ATKS00]. In [KTS02] first results in two dimensions were presented. Three-dimensional results and validation experiments can be found in [KTH<sup>05</sup>].

The simulation of free surfaces clearly requires a distinction between regions that contain fluid and regions that contain only gas. This is done by marking cells that contain no fluid as empty in the flag field. As with obstacle cells, the DFs of these cells are completely ignored during the simulation. However, in contrast to boundary cells, the fluid might at some point in the simulation move into this empty area. To track the fluid motion, another cell type is introduced: the interface cell. These cells form a closed layer, as shown in Figure 4.1. between fluid and empty cells. Here the real work for the simulation and tracking of the free surface is done. It consists of three steps – the computation of the interface movement, the boundary conditions at the fluid interface, and the re-initialization of the cell types. In the next sections, the steps that are executed for an interface cell instead of the standard stream and collide step are described. An overview of the procedure is given in Figure 4.2.

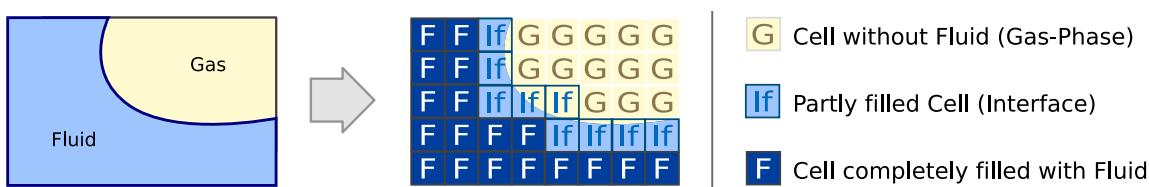


Figure 4.1: Here the different cell types required for the free surface algorithm can be seen.

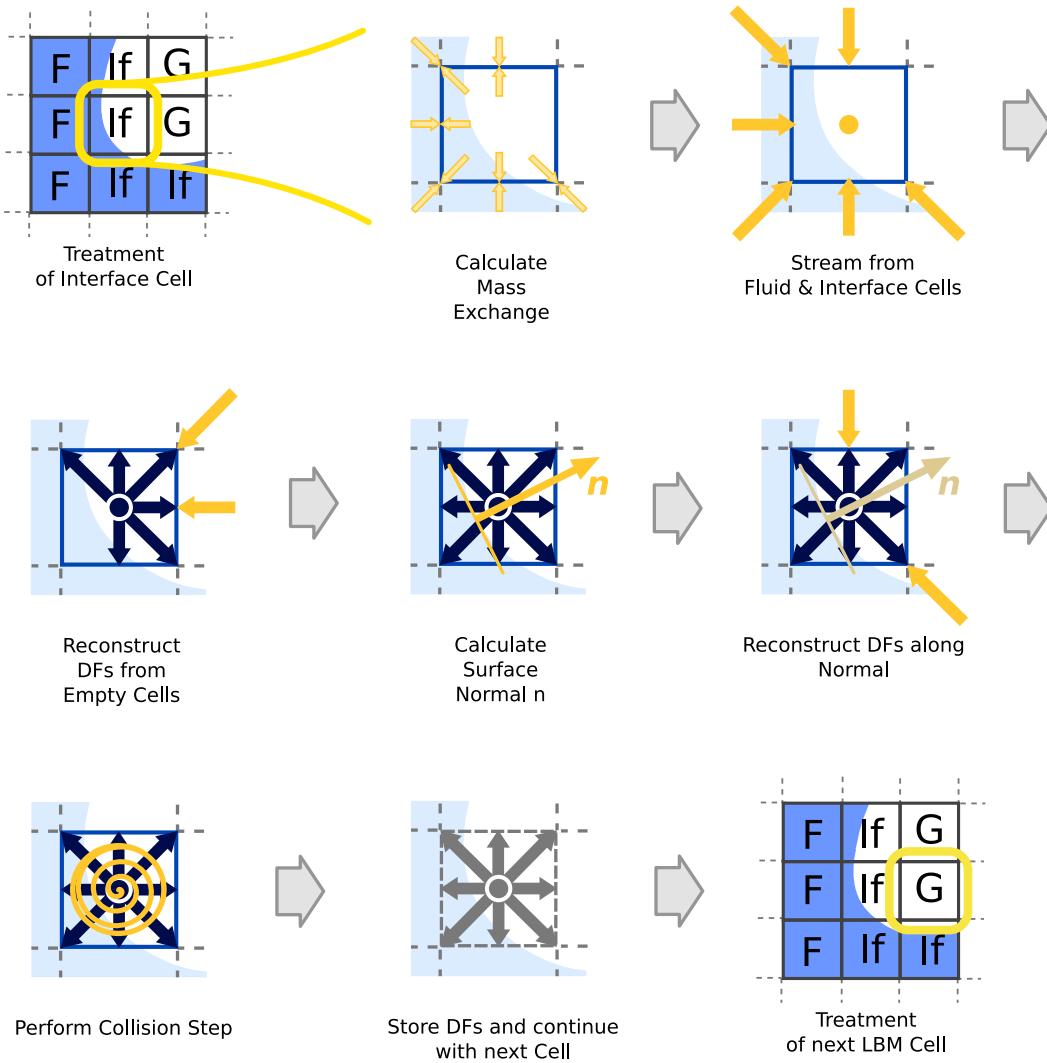


Figure 4.2: An illustration of the steps that have to be executed for an interface cell can be seen in this figure.

## 4.1 Interface Movement

The movement of the fluid interface is tracked by the calculation of the mass that is contained in each cell. This requires two additional values to be stored for each cell: the mass  $m$ , and the fluid fraction  $\epsilon$ . The fluid fraction is computed with the cell mass and density:

$$\epsilon = m/\rho . \quad (4.1)$$

Similar to VOF methods, the interface motion is tracked by computing the fluxes between the cells. However, as the DFs correspond to a certain number of particles, the change of mass is directly computed from the values that are streamed between two adjacent cells for each of the directions in the model. For an interface cell and a fluid cell at  $(\mathbf{x} + \Delta t \mathbf{e}_i)$  this is given by:

$$\Delta m_i(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x} + \Delta t \mathbf{e}_i, t) - f_i(\mathbf{x}, t). \quad (4.2)$$

Table 4.1: Substituting  $s_e$  of Equation (4.3) with the appropriate term given here forces the undesired interface cells to fill or empty. In this table  $\mathbf{x}_{nb}$  denotes the position of the neighboring cell:  $\mathbf{x}_{nb} = \mathbf{x} + \Delta t \mathbf{e}_i$ .

	standard cell at ( $\mathbf{x}_{nb}$ )	no fluid neighbors at ( $\mathbf{x}_{nb}$ )	no empty neighbors at ( $\mathbf{x}_{nb}$ )
standard cell at ( $\mathbf{x}$ )	$(f_{\bar{i}}(\mathbf{x}_{nb}, t) - f_i(\mathbf{x}, t))$	$f_{\bar{i}}(\mathbf{x}_{nb}, t)$	$-f_i(\mathbf{x}, t)$
no fluid nb.'s at ( $\mathbf{x}$ )	$-f_i(\mathbf{x}, t)$	$(f_{\bar{i}}(\mathbf{x}_{nb}, t) - f_i(\mathbf{x}, t))$	$-f_i(\mathbf{x}, t)$
no empty nb.'s at ( $\mathbf{x}$ )	$f_{\bar{i}}(\mathbf{x}_{nb}, t)$	$f_{\bar{i}}(\mathbf{x}_{nb}, t)$	$(f_{\bar{i}}(\mathbf{x}_{nb}, t) - f_i(\mathbf{x}, t))$

The first DF is the amount of fluid that is entering this cell in the current time step, the second one the amount that is leaving the cell. The mass exchange for two interface cells has to take into account the area of the fluid interface between the two cells. It is approximated by averaging the fluid fraction values of the two cells. Thus Equation (4.2) becomes

$$\Delta m_i(\mathbf{x}, t + \Delta t) = s_e \frac{\epsilon(\mathbf{x} + \Delta t \mathbf{e}_i, t) + \epsilon(\mathbf{x}, t)}{2}, \quad (4.3)$$

with  $s_e = f_{\bar{i}}(\mathbf{x} + \Delta t \mathbf{e}_i, t) - f_i(\mathbf{x}, t)$ .

Both equations are completely symmetric, as the amount of fluid leaving one cell has to enter the other one, and vice versa. It thus holds that:  $\Delta m_i(\mathbf{x}) = -\Delta m_{\bar{i}}(\mathbf{x} + \Delta t \mathbf{e}_i)$ . For interface cells with neighboring fluid cells, the mass change has to conform to the DFs exchanged during streaming, as fluid cells don't require additional computations. Their fluid fraction is always equal to one, and their mass equals their current density. The mass change values for all directions are added to the current mass for interface cells, resulting in the mass for the next time step:

$$m(\mathbf{x}, t + \Delta t) = m(\mathbf{x}, t) + \sum_{i=1}^{19} \Delta m_i(\mathbf{x}, t + \Delta t). \quad (4.4)$$

## 4.2 Free Surface Boundary Conditions

As described above, the DFs of empty cells are never accessed. However, interface cells always have empty cell neighbors. Thus, during the stream step only DFs from fluid cells or other interface cells are streamed normally, while the DFs that would be read from empty cells need to be reconstructed with corresponding boundary conditions at the free surface. These boundary conditions do not require additional constructs,

such as ghost layers around the interface. Thus, they can be treated locally for each cell. An atmospheric pressure of  $\rho_A = 1$  is used, as this is also the reference density and pressure of the fluid. Moreover, it is assumed that the viscosity of the fluid is significantly lower than that of the gas phase, while having a higher density. Hence, the gas follows the fluid motion at the interface. In terms of distribution functions, this means that if at  $(\mathbf{x} + \Delta t \mathbf{e}_i)$  there is an empty cell:

$$f_i'(\mathbf{x}, t + \Delta t) = f_i^{eq}(\rho_A, \mathbf{u}) + f_{\bar{i}}^{eq}(\rho_A, \mathbf{u}) - f_i(\mathbf{x}, t), \quad (4.5)$$

where  $\mathbf{u}$  is the velocity of the cell at position  $(\mathbf{x})$  and time  $t$  according to Equation (3.2). The pressure of the atmosphere onto the fluid interface is introduced by using  $\rho_A$  for the density of the equilibrium DFs. Applying Equation (4.5) to all directions with empty neighbor cells would result in a full set of DFs for interface cells. However, to balance the forces on each side of the interface, the DFs coming from the direction of the interface normal are also reconstructed. Thus, if the DF  $f_i$  would be streamed from an empty cell, or if

$$\mathbf{n} \cdot \mathbf{e}_{\bar{i}} > 0 \quad \text{with} \quad \mathbf{n} = \frac{1}{2} \begin{pmatrix} \epsilon(\mathbf{x}_{j-1,k,l}) - \epsilon(\mathbf{x}_{j+1,k,l}) \\ \epsilon(\mathbf{x}_{j,k-1,l}) - \epsilon(\mathbf{x}_{j,k+1,l}) \\ \epsilon(\mathbf{x}_{j,k,l-1}) - \epsilon(\mathbf{x}_{j,k,l+1}) \end{pmatrix} \quad (4.6)$$

holds,  $f_i$  is reconstructed using Equation (4.5). Here  $\mathbf{x}_{j,k,l}$  simply denotes the position of the cell at plane  $l$ , row  $k$  and column  $j$  in the array. Hence, the normal is approximated with central differences of the fluid fraction in each spatial direction.

Now all DFs for the interface cell are valid, and the standard collision is performed with Equation (3.4). The density that was calculated during collision, is now used to check whether the interface cell filled or emptied during this time step:

$$\begin{aligned} m(\mathbf{x}, t + \Delta t) &> (1 + \kappa)\rho(\mathbf{x}, t + \Delta t) \rightarrow \text{cell filled,} \\ m(\mathbf{x}, t + \Delta t) &< (0 - \kappa)\rho(\mathbf{x}, t + \Delta t) \rightarrow \text{cell emptied.} \end{aligned} \quad (4.7)$$

An additional offset  $\kappa = 10^{-3}$  is used instead of 0 or 1 for the emptying and filling thresholds to prevent the new surrounding interface cells from being re-converted in the following LB step. Instead of immediately converting the emptied or filled cells themselves, their positions are stored in a list (one for emptying, another one for filling cells), and the conversion is done when the main loop over all cells has been completed.

### 4.3 Flag Re-initialization

This step takes place when all cells have been updated, and to ensure two properties: the layer of interface cells has to be closed again, once the filled and emptied interface cells have been converted into their respective types. Additionally, the conservation of mass has to be maintained during the conversion. While empty and fluid cells have a mass of exactly zero and one, respectively, interface cells that have filled or emptied according to Equation (4.7) usually have an excess mass on conversion. This excess mass that can be positive or negative needs to be distributed to the neighboring interface cells.

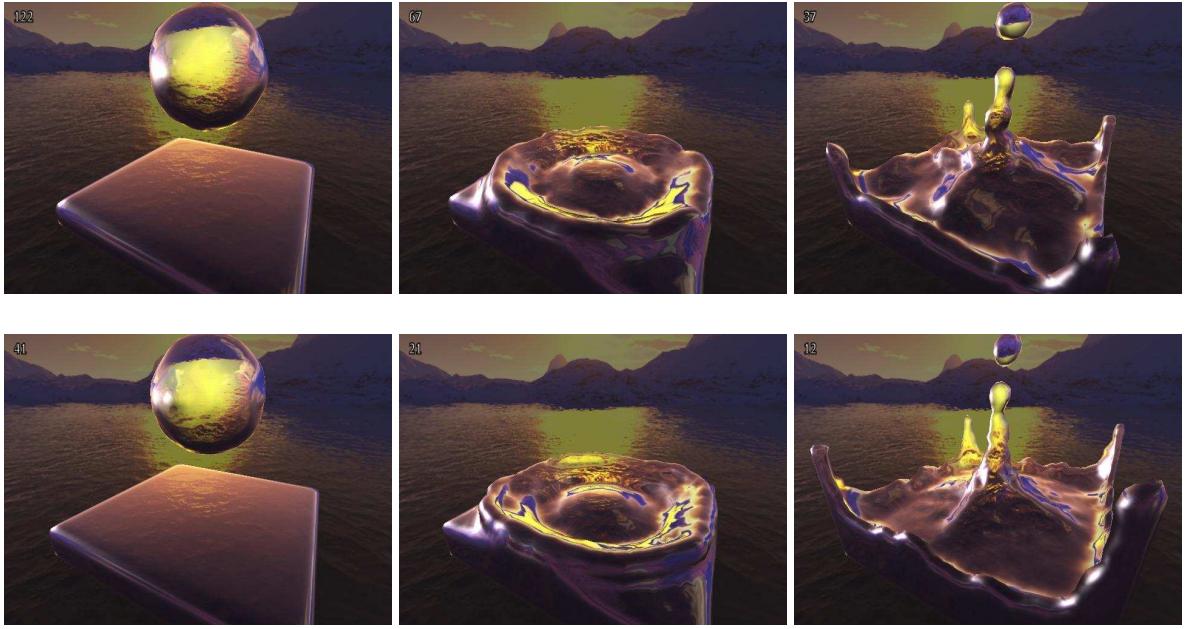


Figure 4.3: Two animations of a single drop falling into a pool of fluid. The upper row of pictures uses a domain size of  $28^3$  running with an average of 76 frames per second, while the lower one uses a resolution of  $38^3$  with an average frame rate of 27.

First the neighborhood of all filled cells is prepared. All neighboring empty cells are converted to interface cells. For each of these the average density  $\rho^{avg}$  and velocity  $\mathbf{v}^{avg}$  of the surrounding fluid and interface cells are computed. The DFs of the empty cells are then initialized with the equilibrium  $f_i^{eq}(\rho^{avg}, \mathbf{v}^{avg})$ . Here it is necessary to remove any interface cells that are needed as boundary for a filled cell from the list of emptied interface cells. During the same pass, the flag of the filled cells is changed to fluid. Likewise, for all emptied cells the surrounding fluid cells are converted to interface cells, simply taking the former fluid cell's DFs for each corresponding new interface cell. Furthermore, the emptied interface cells are now marked as being empty. In a second pass, the excess mass  $m^{ex}$  is distributed among the surrounding interface cells for each emptied and filled cell.  $m^{ex}$  is equal to the mass of the cell  $m$  for emptied cells (according to Equation (4.7) this value is negative), and is calculated as  $(m - \rho)$  for filled cells.

Negative mass values in emptied interface cells, like the mass values larger than the density in filled ones, mean that the fluid interface moved beyond the current cell during the last time step. To account for this, the mass is not distributed evenly among the surrounding interface cells, but weighted according to the direction of the interface normal  $\mathbf{n}$  (which is computed as in Equation (4.6)):

$$m(\mathbf{x} + \Delta t \mathbf{e}_i) = m(\mathbf{x} + \Delta t \mathbf{e}_i) + m^{ex}(\eta_i / \eta_{total}). \quad (4.8)$$

Here  $\eta_{total}$  is the sum of all weights  $\eta_i$ , each of which is computed as

$$\eta_i = \begin{cases} \mathbf{n} \cdot \mathbf{e}_i & \text{if } \mathbf{n} \cdot \mathbf{e}_i > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{for filled cells, and} \quad (4.9)$$

$$\eta_i = \begin{cases} -\mathbf{n} \cdot \mathbf{e}_i & \text{if } \mathbf{n} \cdot \mathbf{e}_i < 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{for emptied cells.}$$

As the mass of the adjacent interface cells changes, the fluid fraction also needs to be changed accordingly. For the steps described so far it is important that they yield the same results independent of the order in which the filled and emptied cells are converted. Thus, the interpolation for empty cells may only interpolate values from cells that aren't new interface cells themselves. Once the cell conversions are complete, the current grid is valid, and is advanced by again starting the main loop over all cells.

## 4.4 Interface Cell Artifacts

The algorithm described so far is already usable to animate free surfaces. However, it can happen that single interface cells are left behind when the fluid moves on, or that interface cells get enclosed in fluid. Although these cases do not perturb the fluid simulation, they are visible as artifacts. To alleviate these problems the following rules are added to the algorithm. The basic idea is to force leftover interface cells without fluid neighbors to empty, and force interface cells without empty neighbors to fill. This is done by substituting the term  $(f_i(\mathbf{x} + \Delta t \mathbf{e}_i, t) - f_i(\mathbf{x}, t))$  from Equation (4.3) according to Table 4.1. In rare cases, where these cells still remain interface cells, they are simply treated as filled or emptied cells. Thus, if a cell has no fluid neighbors and its mass drops below  $(0.1 \cdot \rho)$ , it is treated as having emptied in this time step. Likewise cells with no fluid neighbors and a mass larger than  $(0.9 \cdot \rho)$  are treated as having been filled. In contrast to conventional VOF methods this makes it unnecessary to, e.g., add an artificial surface tension in order to ensure a proper interface cell layer.

## 4.5 Interactive Simulations

The capabilities of the algorithm will be shown using four different setups, two with obstacles, and two without. All four use values of  $\omega$  between 1.85 and 1.95. Note that these simulations do not make use of the turbulence model of Section 3.3, and thus only rely on the adaptive time stepping, explained in more detail in Section 6, to ensure stability. The pictures are screenshots from real-time calculations performed on a standard Pentium 4 CPU (Northwood core) with 3.0 GHz, 512KB Level 2 Cache, and a state of the art graphics card. The latter, however, was not a limiting factor for the shown test cases. The number of frames per second can be seen in the upper left corner of each picture. It includes the calculation of the fluid movement and the visualization of the surface using a marching cubes algorithm [LC87]. We simply use the fluid fraction values  $\epsilon$  and triangulate the isosurface at  $\lambda = 1/2$ . This surface generation was already used in [Thu03], among others. As the values of  $\epsilon$  are cut off at 0 or 1 for empty

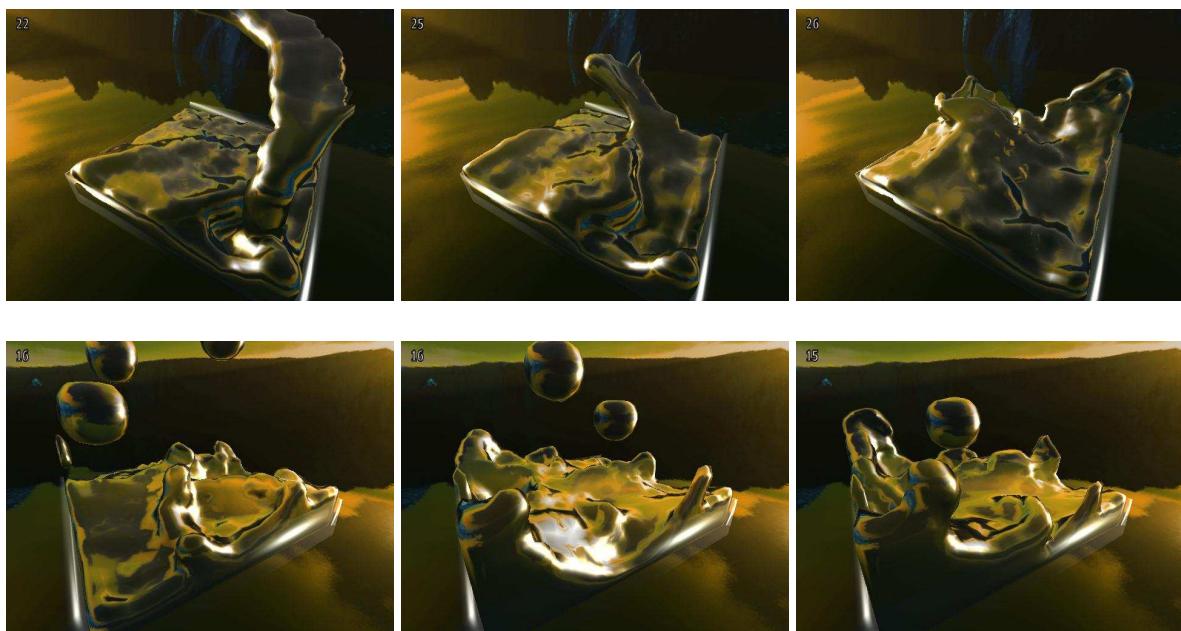


Figure 4.4: A stream of fluid and two drops, in the top and bottom row, respectively, hit a rectangular container partly filled with fluid. Both are screenshots from the sample application that allows the user to interactively place the drops.

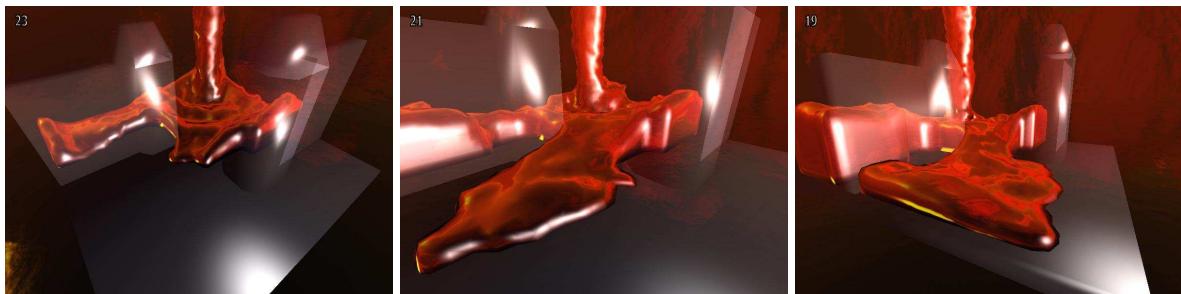


Figure 4.5: A stream of fluid fills up a Z-shaped domain, with an average frame rate of 26. In the end more than 11000 cells of the  $36^3$  grid are filled with fluid.

or fluid cells, respectively, we perform a filtering step before triangulation to acquire more accurate normals and a smoother appearance of the fluid surface. Including the triangulation the whole visualization requires ca. 10% of the total computational work for the shown animations.

The screenshots of Figure 4.3 are from a test case where a single drop is falling into the center of a pool of liquid. For the bottom row of pictures the size of the computational domain is  $38^3$ , with on average more than 10000 interface or fluid cells. This number, together with the occurring maximum velocities, determines the overall performance. The animation of Figure 4.3 runs with an average frame rate of 27, which drops to 11 once the waves from all 4 corners of the domain splash together in the middle, resulting in high upward velocities. Calculating the same animation with a

resolution of  $28^3$  (top row of Figure 4.3) and on average more than 4000 used cells results in a similar fluid motion. In this case the minimal and average frame rate are 35 and 76, respectively. For the  $38^3$  case, the simulation itself with 2500 LB steps takes 16.4 seconds on the Pentium 4 system. Using an Athlon 64 4000 with 2.4 GHz and 1MB Level 2 Cache, the same calculations can be performed in 13.1 seconds. Depending on the current ratio of interface and fluid cells, the implementation can handle more than 2 million cell updates per second, and up to 3 million updates on the Athlon 64 system. To gain this performance, it is important to optimize the flag array tests, and unroll loops over the 19 distribution functions for standard fluid cells.

Screenshots from the interactive demo application are shown in Figure 4.4. Here a user can paint drops or lines of fluid into the domain with the mouse, resulting in turbulent and chaotic flow patterns. In both cases the average frame rate drops towards the end of each animation, when around 10000 cells are used. However, with frame rates between 20 and 30 the application remains very responsive.

Figure 4.5 and 4.6 show animations with obstacles in the domain. In the first case, fluid is poured into one corner of a Z-shaped domain, filling it with fluid. The second example is again from an interactive program run, and shows drops of fluid falling into a bowl shaped obstacle in the middle of the domain. Especially the latter case results in very complex flows. Here the size of the domain is  $44^3$  with up to 6000 cells being filled with fluid. The average frame rate is over 40 as the fluid always comes to rest between the subsequent drops. The frame rate is reduced when high velocities are caused by up to 15 drops hitting the obstacle simultaneously. The simulation, however, remains stable due to the adaptive time stepping algorithm of Section 6.

The previous examples show that this basic free surface LBM yields efficient and stable simulations. For resolutions as those used for the animations presented in this section, the method achieves interactive frame rates. The method is inherently mass conserving, and typical artifacts of VOF methods can be overcome at the price of a slightly reduced accuracy. Note that the computational efficiency of a single LB step means that all additional computations for each step have to be similarly fast. A full level set initialization and advection, as demonstrated in [TR04], would for example significantly reduce the efficiency.

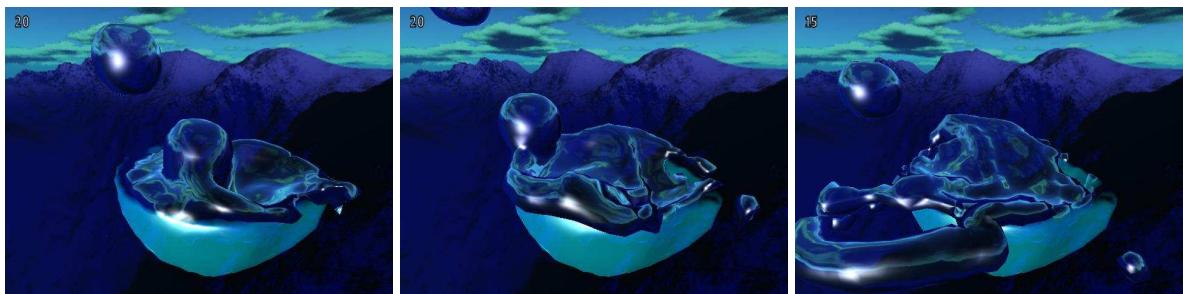


Figure 4.6: Several interactively placed drops of fluid hit a bowl-shaped obstacle, resulting in complex splashes.





It's down there somewhere, let me take another look.  
J. Lebowski, the Big Lebowski

# Chapter 5

## Moving Obstacles

In order to simulate interactions of objects with a fluid, the simulation has to at least handle moving boundary conditions. Hence, the fluid has to be correctly accelerated and pushed away at the obstacle surface. This represents a one-way coupling between obstacle and fluid, as forces from the object are transferred to the fluid. If the motion of the object is itself determined by another simulation, e.g. to handle rigid body collisions or buoyancy effects, a two-way coupling is necessary to also compute the forces acting on the immersed body due to the fluid flow. The following sections will explain how to include one- and two-way coupled moving obstacles into LB free surface simulations.

For other popular fluid simulation algorithms, the treatment of moving and deforming obstacles was handled in a variety of ways. In [MST<sup>+</sup>04] the authors couple an SPH solver with tetraeder meshes and an finite element solver. For level set based NS solvers, objects can either be treated as a rigid type of fluid [CMT04], or with a method proposed in [GSLF05] that is also able to handle thin shells. Another solver type that works on directly on dynamically changing meshes is described in [KFCO06]. This approach allows a finer mesh around moving objects, to increase the accuracy of the simulation. A similar effect can be achieved for the LBM with the technique described in Section 7.

---

## 5.1 Obstacle Boundary Conditions

The standard way to implement obstacles for LB fluid simulations is the *bounce-back* scheme for no-slip boundary conditions. They result in a zero normal and tangential velocity at the obstacle boundary. In terms of DFs this means that during the streaming step the DFs are reflected at the obstacle surface, thus Equation (3.1) is changed to

$$f_i(\mathbf{x}, t + \Delta t)' = f_{\tilde{i}}(\mathbf{x}, t) \quad (5.1)$$

for all cells where the neighbor along velocity vector  $\mathbf{e}_i$  is an obstacle cell. Note that  $f_{\tilde{i}}$  denotes the DF along the inverse velocity vector of  $f_i$ , thus  $\mathbf{e}_{\tilde{i}} = -\mathbf{e}_i$ . These boundary conditions can be used to model "sticky" walls that slow down the fluid – e.g. rough stone surfaces.

Another type of boundary conditions are free-slip boundary conditions, which only result in a normal velocity of zero, but leave the tangential velocity at the obstacle interface unchanged. Hence, during the streaming step of the LBM, the DFs are reflected only along the obstacle normal, in contrast to Equation (5.1), where they are reflected along normal and tangential direction. This free-slip scheme is shown to the right of Figure 5.1. Note that while no-slip boundary conditions can be handled locally for a cell, free-slip conditions require DFs from a neighboring cell. Furthermore, the free-slip handling is equivalent to the no-slip boundaries if the neighboring cell in the tangential direction is not a fluid cell. Free-slip boundary conditions can be used to model smooth surfaces that do not slow down the fluid, such as glass walls.

To model materials that have properties in between the two extremes described above, the free- and no-slip boundary conditions can be linearly interpolated. Given the reflected DFs  $f_r$  from the free-slip treatment, this gives:

$$f_i(\mathbf{x}, t + \Delta t)' = w_p f_{\tilde{i}}(\mathbf{x}, t) + (1 - w_p) f_r(\mathbf{x}, t), \quad (5.2)$$

where  $w_p$  is the parameter to control the surface smoothness. For  $w_p = 1$  Equation (5.2) reduces to no-slip boundary conditions, while  $w_p = 0$  yields a free-slip boundary.

The boundary conditions here have first order accuracy (for arbitrary shapes). Higher order boundary conditions that take into account the position of the obstacle surface along each lattice connection, the most common second order one being [BdLL01], can also be applied to the LBM. These, however, would significantly increase the computational overhead for moving obstacles, and Section 5.5 will demonstrate that the first order boundary conditions already give good results.

## 5.2 Moving Boundary Conditions

If the obstacle is moving, the momentum of the movement has to be transferred to the fluid, as described in [Lad94]. While the basic no-slip handling from Equation (5.1) remains the same, an additional forcing term is added during streaming:

$$f_i(\mathbf{x}, t + \Delta t)' = f_{\tilde{i}}(\mathbf{x}, t) + 2 w_i \rho_f 3 \mathbf{e}_i \cdot \mathbf{u}_o, \quad (5.3)$$

where  $\rho_f$  is the fluid density and  $\mathbf{u}_o$  the obstacle velocity at the obstacle boundary. The fluid density can be approximated by the initial density, hence  $\rho_f = 1$ .

For free-slip boundary conditions, the acceleration should only occur in the normal direction of the obstacle surface. The velocity to be used in Equation (5.3) is thus projected onto the surface normal. The forcing term is now only added to those DFs where the reflection is reduced to Equation (5.1). This is due to the fact that tangential to the free-slip surface there is no acceleration of the fluid due to the surface smoothness. Thus the fluid adjacent to a free-slip obstacle only needs to be accelerated along the normal direction. To still keep the flux balance for free-slip boundaries, the acceleration term of Equation (5.3) needs to be multiplied by 2. For free- and part-slip boundary conditions Equation (5.3) is thus changed to

$$\begin{aligned} f_i(\mathbf{x}, t + \Delta t)' &= w_p \left( f_{\bar{i}}(\mathbf{x}, t) + 2 w_i \rho_f 3 \mathbf{e}_i \cdot \mathbf{u}_o \right) + (1 - w_p) \\ &\quad \left( f_r(\mathbf{x}, t) + w_r(\mathbf{x}, t) 2 w_i \rho_f 3 \mathbf{e}_i \cdot [(\mathbf{n}_o \cdot \mathbf{u}_o) \mathbf{n}_o] \right), \end{aligned} \quad (5.4)$$

where  $\mathbf{n}_o$  is the obstacle normal at  $\mathbf{x}$ .  $w_r$  is an indicator function that has a value of one if the streaming of  $f_r$  reduces to a no-slip reflection, and zero otherwise. The forcing term is thus only applied during the stream step for DFs such as the center arrow of the source cell in Figure 5.1 near free-slip boundaries.

While this deals with the obstacle to fluid momentum transfer, the force acting on the obstacle due to the fluid pressure and movement can be calculated as

$$\begin{aligned} F_o &= \frac{\Delta x}{\Delta t} \sum_{\forall \mathbf{x}_b} \sum_{i=1}^{19} \mathbf{e}_i w_o(\mathbf{x}_b + \mathbf{e}_i \Delta t) \\ &\quad \left( f_{\bar{i}}(\mathbf{x}_b + \mathbf{e}_i \Delta t, t) + f_i(\mathbf{x}_b + \mathbf{e}_i \Delta t, t + \Delta t) \right), \end{aligned} \quad (5.5)$$

where  $w_o(\mathbf{x})$  is an indicator function that is equal to one when the cell at  $\mathbf{x}$  is a fluid cell, and zero otherwise. Here the first sum traverses all boundary cells with position  $\mathbf{x}_b$  of the obstacle. Equation (5.5) thus approximates the surface integral of the shear stresses and pressure forces along the obstacle surface. Hence, with Equation (5.3) and Equation (5.4) the fluid to obstacle coupling is computed, while the combination with Equation (5.5) enables full two-way coupled fluid simulations.

## 5.3 Lattice Initialization

Triangular meshes were used to specify the obstacle volumes. For the simulation, the movement and position information has to be transferred from the triangle mesh to the LB grid. This is done by creating point samples from the triangle mesh that ensure a closed layer for the obstacle surface and can furthermore be used to generate velocity information for the boundary conditions.

### Point Samples

The point samples on the mesh surface are not required to be regularly spaced, but have to ensure that a closed layer of obstacle cells is created for a given LB grid resolution. No fluid element is allowed to move from one side of a thin obstacle to the other.

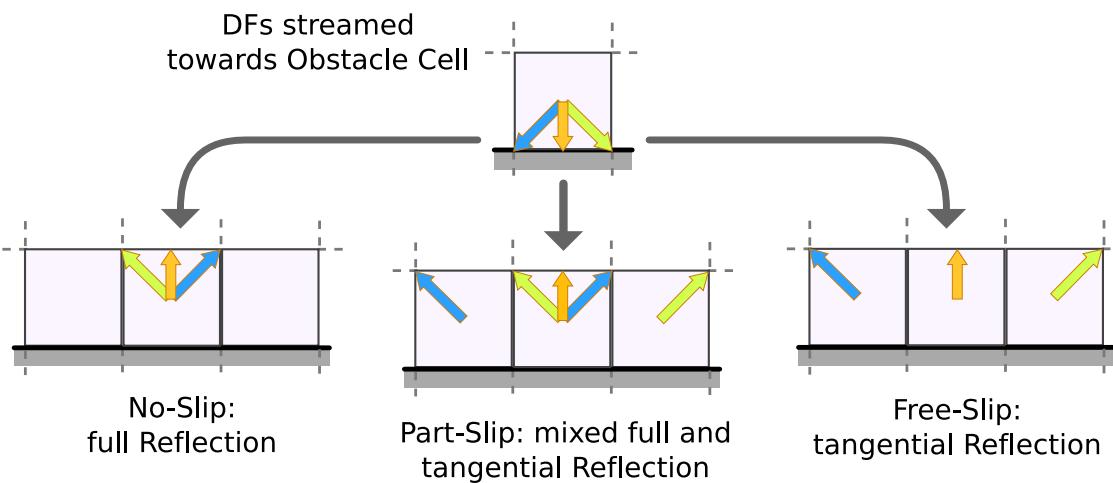


Figure 5.1: The different types of boundary conditions for LB obstacles.

Thus, no two fluid cells on opposing sides of an obstacle should be connected by a lattice velocity vector. Depending on the size of a grid cell  $\Delta x$  a feature size  $s_f = \Delta x/2$  is used in the following. Given a triangle with points  $p_1, p_2, p_3$  and normal  $n$ , the number of divisions along the sides are computed as

$$s_u = \text{floor}\left(\frac{|p_2 - p_1|}{s_f}\right), \quad s_v = \text{floor}\left(\frac{|p_3 - p_1|}{s_f}\right). \quad (5.6)$$

Then the points  $o$  on the surface can be computed using barycentric coordinates:

$$\mathbf{o}_{u,v,1}, \mathbf{o}_{u,v,2} = \left(1 - \frac{u + 1/4}{s_u} - \frac{v + 1/4}{s_v}\right)\mathbf{p}_1 + \frac{u + 1/4}{s_u}\mathbf{p}_2 + \frac{v + 1/4}{s_v}\mathbf{p}_3 \pm \frac{1}{4}s_f\mathbf{n}, \\ \text{with } 0 \leq u \leq s_u, 0 \leq v \leq s_v, u + v \leq 1. \quad (5.7)$$

Note that Equation (5.7) creates two points, each of which are offset by the triangle normal and feature size. Equation (5.6) guarantees that the points have a sufficiently small spacing in a plane of the grid, while the normal offset of Equation (5.7) ensures that the obstacle layer has a thickness of one to two cells.

To ensure that a minimum of points is generated, the points  $p_1, p_2$  and  $p_3$  of the triangle are permuted to ensure that the values  $s_u$  and  $s_v$  of Equation (5.6) are minimal. Hence, the two shortest sides of the triangle are used for point generation.

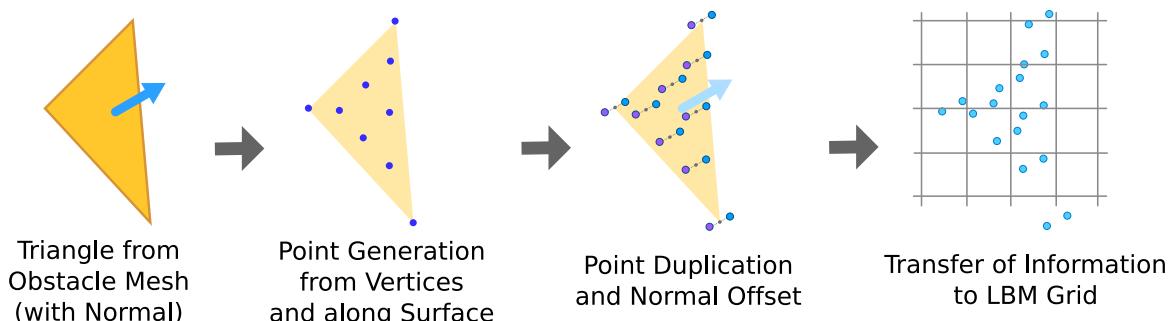


Figure 5.2: Overview of the obstacle initialization for a single triangle of the mesh.

## Grid Initialization

The whole collection of points  $\mathcal{P}$  for an obstacle object thus contains the points generated by Equation (5.7) for all triangles, and the original vertices of the mesh. The vertices are also offset by  $\pm 1/4s_f\mathbf{n}$ , and are necessary as  $s_u$  and  $s_v$  can be zero. This would mean that no points are generated in the plane of the triangle, but as the triangle is smaller than a single grid cell, its vertices suffice to initialize the LB grid. This, however, also means that if the mesh is finely triangulated, not all of the vertices would be necessary for initialization. In this case an external program or suitable software library could be used to simplify the mesh, possibly according to the feature size. In general, the method described above assumes that the feature size is usually significantly smaller than the average triangle size, which is normally the case for detailed fluid animations.

For animated meshes it is required that the animation is described by movements of the vertices, thus the triangle structure itself doesn't change. Before each time step of the LBM, the point set  $\mathcal{P}(t)$  is generated for the current time step. Furthermore either the point set of the last time step  $\mathcal{P}(t - \Delta t)$  can be used, or generated anew. Now a loop over all points  $\mathbf{p}(t)_i$  in  $\mathcal{P}(t)$  yields the grid positions of obstacle cells to be initialized for the current time step. The obstacle velocity  $\mathbf{u}$  for Equation (5.1) can be computed with

$$\mathbf{u} = (\mathbf{p}(t)_i - \mathbf{p}(t - \Delta t)_i) / \Delta x . \quad (5.8)$$

For optimization purposes, the acceleration term of Equation (5.1) can be precomputed and stored in the LB grid, as obstacle cells do not require DFs to be stored. As often more than a single point of  $\mathcal{P}$  maps to one grid cell, these multiple values could be averaged with appropriate weights. However, as initializing a cell with the first point that maps to it yields good results – this scheme will be used in the following.

During this pass over the obstacle points the maximum velocity on the obstacle surface is easily computed, and can be used to adapt the time step size of the simulation correspondingly. In a second pass of the point set  $\mathcal{P}(t - \Delta t)$  old obstacle cells have to be removed from the grid. These can be initialized by empty cells, or, if a fluid cell is in the neighborhood with an interface cell. The latter case is necessary as fluid cells are not allowed to be in the neighborhood of an empty cell, so alternatively the fluid cell could be converted to an interface cell.

## Mass Conservation

Note that the boundary conditions so far do not conserve mass – due to the approximation of the obstacle boundary with cells, the moving object can cover a different number of cells during its movement. For flows without a free surface, hence with completely submerged objects, this is usually unproblematic. However, with free surface flows, this can lead to significant errors in the mass conservation. To alleviate this problem, the change of mass in the system due a moving no-slip obstacle can be directly computed as

$$\Delta M_o = M_{\text{add}} - M_{\text{sub}} + \sum_{\forall \mathbf{x}_b} 2 w_i \rho_f 3 \mathbf{e}_i \cdot \mathbf{u}_o . \quad (5.9)$$

Here  $M_{\text{add}}$  denotes the gained mass from interface cells that are newly created in the current LB step due to the object movement (as explained in the previous paragraph). On the other hand,  $M_{\text{sub}}$  is the mass that was lost from all fluid cells that were reinitialized as new boundary cells during the last step. The third component is the sum of all acceleration terms from Equation (5.1). Note that Equation (5.9) assumes no-slip boundary conditions, and thus has to be changed as explained above for free- or part-slip obstacles. By accumulating  $M_o(t + \Delta t) = M_o(t) + \Delta M_o$  for each moving obstacle in the simulation, the change of mass due to this object can be calculated. A parameter that can be freely chosen is the fill fraction  $\epsilon_n$  for the newly created interface cells from the pass over the last object position with  $\mathcal{P}(t - \Delta t)$ . We now set  $\epsilon_n = 0$  if  $M_o(t) \geq 0$ , and likewise  $\epsilon_n = 1.5$  when  $M_o(t) < 0$ . As the front side of the object removes mass from the system, the back side can thus be used to control the mass loss. Note that if further accuracy of the mass correction is necessary, the initialization of all  $\epsilon_n$  could be normalized by the number of new interface cells. This, however, would require a second pass over these cells.

## 5.4 Surface Generation

While the fluid can usually be reconstructed directly from the fluid fraction values of the free surface tracking, the surfaces adjacent to obstacles require additional treatment. As described in the previous chapter, the fluid surface itself is given by the fill fraction isolevel  $\lambda = 1/2$ . The obstacle initialization as described above will result in a fluid surface around the obstacle surface without touching it. Thus a separation of fluid and obstacle surfaces is visible, in contrast to a real fluid that would connect with the immersed object at the interface and not have a visible surface in the immersed regions.

This appearance can be achieved by extrapolating the fluid fraction values into the

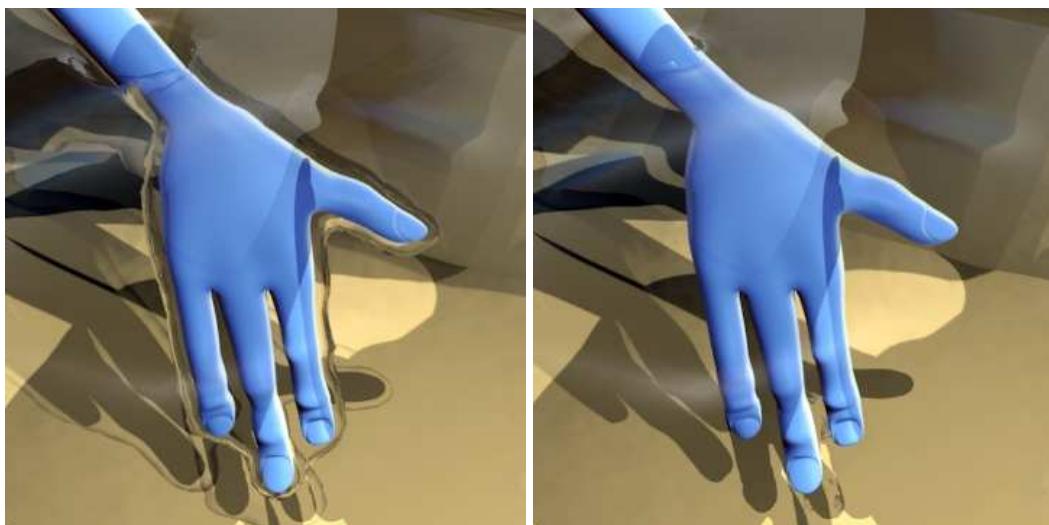


Figure 5.3: Comparison of the two surface generation approaches. To the left, surfaces are generated for domain sides and around the submerged obstacle. To the right, these surfaces are removed with the algorithm described in Section 5.4.

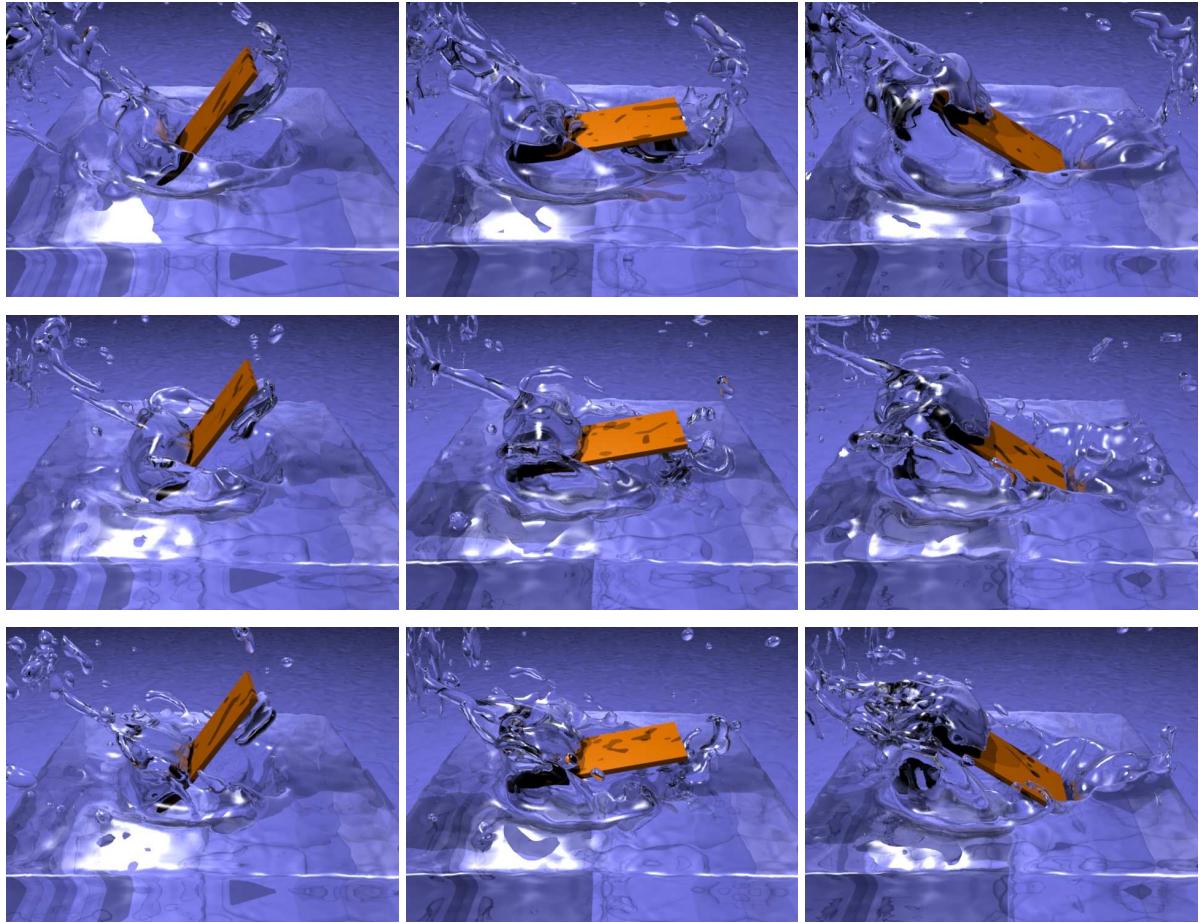


Figure 5.4: Test case for moving rigid bodies: a rotating box is lowered into a basin of fluid. The upper row of pictures uses no-slip boundary conditions for the box, the middle one part-slip and the lowest one free-slip boundary conditions. An increased amount of splashes is visible in the lower rows of pictures.

obstacle layer. We thus set fluid fraction values for obstacle cells  $\epsilon_b$  in the following way:  $\epsilon_b(\mathbf{x}) = 1$  is used for all obstacle cells at  $\mathbf{x}$  with fluid, but without interface cell neighbors. If an obstacle cell has both fluid and interface neighbors, its fluid fraction value is set to an average fluid fraction with

$$\epsilon_b(\mathbf{x}) = \frac{\sum_{i=1}^{19} w_b(\mathbf{x} + \mathbf{e}_i \Delta t) \epsilon_b(\mathbf{x} + \mathbf{e}_i \Delta t)}{\sum_{i=1}^{19} w_b(\mathbf{x} + \mathbf{e}_i \Delta t)}, \quad (5.10)$$

where  $w_b(\mathbf{x})$  is an indicator function that is one if the cell at  $\mathbf{x}$  is either a fluid, interface or empty cell, and zero otherwise. In a second pass, all obstacle cells that have not been modified, and thus are not in the neighborhood of the fluid, are set to  $\epsilon_b = 0.99\lambda$ . This is necessary as the initialization from Section 5.3 can result in two layers of obstacle cells, in which case the surface should be moved further inwards to ensure that is inside of the original mesh.

An example of this modified surface generation can be seen in Figure 5.3. To the left of Figure 5.3 an image of the uncorrected fluid surface is shown, while the right side shows the actual moved fluid surface. Note that for transparent objects it will

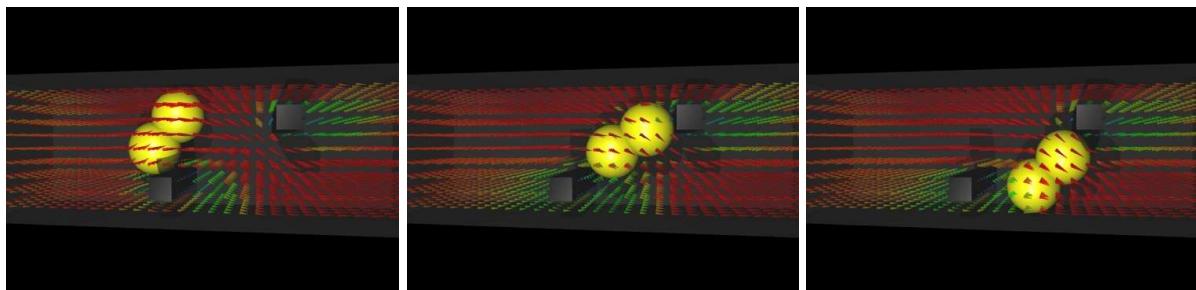


Figure 5.5: Two-way coupling of the boundary conditions. An object consisting of two connected spheres is bouncing through a channel filled with fluid. The arrows indicate the fluid velocity. This simulation was performed by Klaus Iglberger, more information can be found in [Igl05].

moreover be necessary to clip the fluid surface with the obstacle mesh to hide the inner parts while retaining an exact connection of the fluid surface with the obstacle.

## 5.5 Results

In the following three different simulations for the algorithm described above will be presented. The first, and relatively simple, test case with a rigid moving object can be seen in Figure 5.4. Here a simplified "mixer" is modelled – a box is lowered into the fluid while rotating along the  $y$  axis. This simulation was performed with a resolution of  $128^3$  and a viscosity of water. The simulation took 27 seconds per frame on average. This time per frame, as well as the following timings of this section, were measured on a standard workstation with a Pentium4 CPU (3.0GHz). Each row of pictures uses different boundary conditions for the obstacle. From top to bottom the boundary conditions are: no-slip, part-slip with  $w_p = 0.002$  and free-slip. As the effect of  $w_p$  is highly non-linear, the weight has to be small in order to show a noticeable effect of the free-slip boundary conditions. It can be seen that the fluid is correctly accelerated and pushed away by the rectangular shape. The free-slip simulation exhibits a larger amount of splashes, as there is less slowdown in tangential direction. As expected, the part-slip boundary conditions lie in between the two other cases.

A different setup to test the two-way coupling of the moving boundary conditions is shown in Figure 5.5. Here a moving object consisting of two connected spheres is pushed through a channel with fluid and two fixed rectangular obstacles. The object is accelerated up to the fluid velocity, then bounces through the two fixed obstacles. In this case a simulation resolution of  $60 \times 30 \times 30$  was used, with no-slip boundary conditions for all obstacles. Due to the small size, the total simulation only took ca. 15 seconds. The boundary conditions for deforming meshes are demonstrated in Figure 5.6, where an animated character is interacting with the fluid. The simulation with a resolution of  $166 \times 166 \times 200$  took on average 244 seconds per frame, and the obstacle initialization with more than  $125k$  points for the 5038 triangle mesh ca. 7% of the time for each LB step.

The method thus efficiently extends a free surface LB simulation to handle moving and deforming obstacles. The boundary handling furthermore ensures a closed

obstacle cell layer for thin shells. In the future it would be interesting to couple the simulation to a full rigid body simulator, and handle additional effects such as objects with phase transitions [MKN<sup>+</sup>04, LIG06] or even fracturing [PKA<sup>+</sup>05].

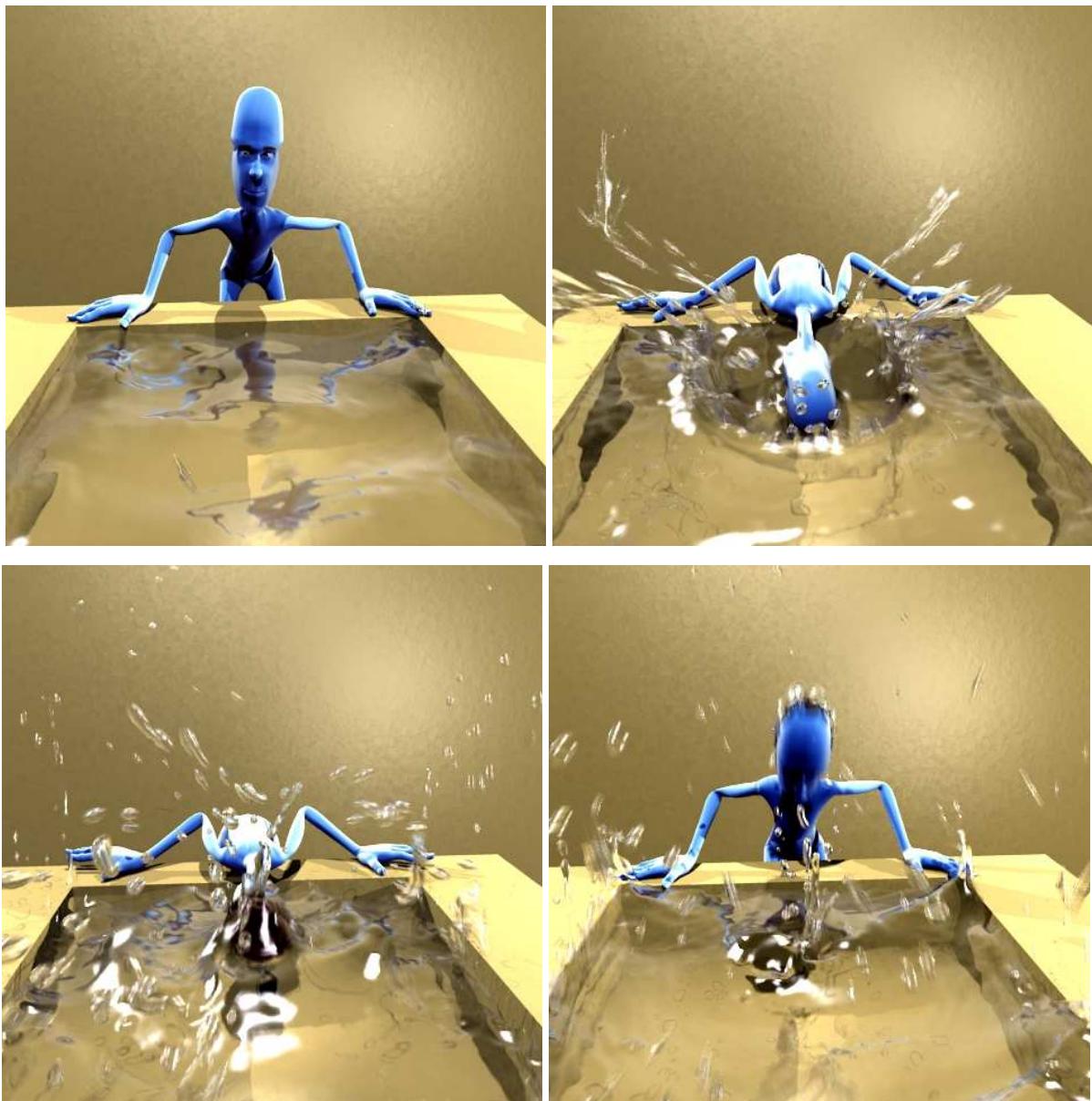


Figure 5.6: Example of a deforming mesh from an animated character interacting with the fluid.



$$g_n = g_o s^2$$

$$\rho_n = (\rho_o - \rho_{\text{ref}})s + \rho_{\text{ref}}$$

$$u_n = u_o s$$

$$s_\tau = \Delta t_n \tau_n / \Delta t_o \tau_o ,$$

$$s_{f_i} = f_i^{eq}(\rho_n, u_n) / f_i^{eq}(\rho_o, u_o) .$$

$$f'_i = [f_i^{eq}(\rho_o, u_o) + (f_i - f_i^{eq}(\rho_o, u_o)) s_\tau] s_{f_i}$$

If rationalism/reductionism is our religion,  
then time keeping is our benediction.  
Buried Inside, Time As Surrogate Religion

# Chapter 6

## Adaptive Time Steps

The stability of the turbulence model from Section 3.3 is transferred directly to the free surface simulations, hence enabling the computation of free surface flows with high Reynolds numbers, and values of  $\tau$  close to 0.5. A remaining source of instability, however, is the problem of lattice velocities becoming too large during the course of the simulation. The method that will be presented in this section can be used to speed up and stabilize these simulations. It uses a reparametrization of the LB simulation that effectively changes the time step size and Mach number. After an explanation of the algorithm, it will be validated by accuracy measurements for two-dimensional and three-dimensional simulations. Examples of the speed up that can be achieved by adaptive time steps will moreover be given.

### 6.1 Adaptive Parametrizations and Mach Numbers

Simulation configurations for free surface simulations usually exhibit a highly varying range of velocities. As the LBM intrinsically limits the maximum velocity of a cell, a setup like this usually requires a time step that is sufficiently small to assure that the largest occurring velocities remain valid during the course of the simulation. To alleviate this restriction, a technique to adaptively modify the parameterization to change the time step size will be presented in the following. This also results in a change of the Mach number  $Ma = \mathbf{u}/c_s$ , as the grid size and thus the speed of the information

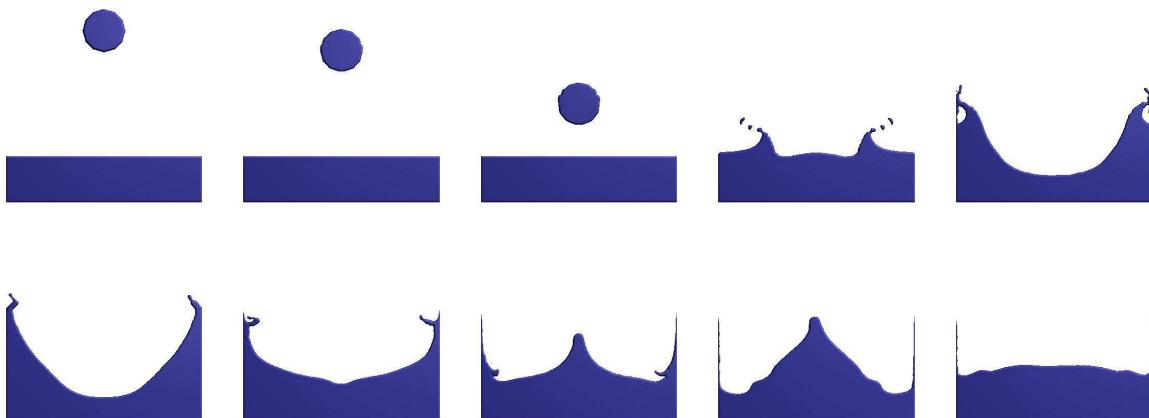


Figure 6.1: Here several frames of the simulation setup for the second test case from Section 6.2 can be seen. A drop of fluid falls into resting fluid in a square computational domain. In this case the grid resolution is  $256^2$ .

propagation (determined by  $c_s$ ) stay fixed, but the velocities  $\mathbf{u}$  are rescaled. However, it will be shown in Section 6.2 that this does not significantly disturb simulations such as gravity driven free surface flows. Other problems, however, that might require a fixed Mach number will not be able to make use of this approach. Given a fluid viscosity  $\nu$  and an acceleration force  $\mathbf{g}$  a maximum time step size  $\Delta t_{\max}$  can be calculated with Equation (3.9) On the other end, the minimum size of a time step is limited by the stability of the LBM. For a BGK model  $\tau_{\min} \geq 0.51$  was chosen. Given the kinematic viscosity of the fluid  $\nu$  and  $\tau_{\min}$ , the lower limit of the time step can be computed as

$$\Delta t_{\min} = \frac{2\tau_{\min} - 1}{6\nu} . \quad (6.1)$$

However, with the use of the turbulence model from Section 3.3 the limitations of the BGK model are overcome. In this case,  $\Delta t_{\min} = 0$  can be used, as the turbulence model compensates the stability problems of small  $\tau$  values. Thus, there is effectively no limit to decrease the time step size.

During each LB step, the current maximum velocity  $\mathbf{u}_{\max}$  is computed. As a threshold for the velocity  $|\mathbf{u}_{\text{thresh}}| = 1/6$  is used, which is half of the speed of sound chosen for D2Q9 and D3Q19. The speed of sound thus represents the upper limit of the lattice velocity to simulate weakly compressible flow. If  $\mathbf{u}_{\max}$  exceeds this threshold during the course of the simulation, the following procedure to stabilize the simulation again is performed. If the velocities become too large, this will result in a smaller time step size, while for small  $\mathbf{u}_{\max}$  the parameters are changed to have the simulation perform larger time steps. The size of the new time step can be determined by

$$\Delta t_n = |\mathbf{u}_{\text{thresh}}| / |\mathbf{u}_{\max}| . \quad (6.2)$$

In the following, a subscript of  $o$  will denote values before the rescaling procedure, while  $n$  will denote values for the new parameterization. Hence,  $t_o$  is the size of the time step previously used for the LBM. To account for the new time step size, the hydrodynamic moments as well as the acceleration force have to be rescaled by  $s =$

$\Delta t_n / \Delta t_o$ :

$$\begin{aligned}\rho_n &= (\rho_o - \rho_{\text{ref}})s + \rho_{\text{ref}} \\ \mathbf{u}_n &= \mathbf{u}_o s \\ \mathbf{g}_n &= \mathbf{g}_o s^2\end{aligned}\quad (6.3)$$

Here  $\rho_{\text{ref}}$  denotes the current reference density of the simulation, which has to be calculated from the total fluid volume  $V$  and the overall mass  $M$  as  $\rho_{\text{ref}} = V/M$ .  $V$  is calculated by the sum of all fluid fraction values  $\epsilon$  (which are equal to one for fluid cells), while  $M$  is computed as the sum of all cell masses. Thus for interface cells  $m$  and for fluid cells  $\rho$  is used. The deviation of the cell densities from the reference density needs to be rescaled, as the changed force  $\mathbf{g}_n$  requires an adaption of the density gradient. This rescaling furthermore changes the values of  $m$  and  $\epsilon$  for interface cells:

$$\begin{aligned}m_n &= m_o(\rho_o / \rho_n), \\ \epsilon_n &= m_n / \rho_n.\end{aligned}\quad (6.4)$$

Since the relaxation time depends on the size of the time step, the non-equilibrium parts of the distribution functions have to be rescaled, similar to the rescaling for simulations using differently refined grids as in Section 7 or [FH98, CKTR03]:

$$f'_i = [f_i^{eq}(\rho_o, \mathbf{u}_o) + (f_i - f_i^{eq}(\rho_o, \mathbf{u}_o)) s_\tau] s_{f_i}, \quad (6.5)$$

where  $s_\tau$  and  $s_{f_i}$  are calculated as:

$$\begin{aligned}s_\tau &= \Delta t_n \tau_n / \Delta t_o \tau_o, \\ s_{f_i} &= f_i^{eq}(\rho_n, \mathbf{u}_n) / f_i^{eq}(\rho_o, \mathbf{u}_o).\end{aligned}\quad (6.6)$$

Here the factor  $s_\tau$  corresponds to the non-equilibrium scaling factor from [FH98], while the additional scaling by  $s_{f_i}$  is necessary to account for the changes of velocity and density.  $s_\tau$  is thus equal for all cells and lattice directions  $i$ . The factor  $s_{f_i}$ , on the other hand, depends on  $i$  and can be different for each cell. The scaling by  $s_{f_i}$  is required to make the DFs of the cell represent the velocity  $\mathbf{u}_n$  and deviation from the median density  $\rho_n$  required by the new parameterization.

Instabilities due to  $\tau$  being almost 0.5 are alleviated by applying the turbulence model. This in turn requires a modification of Eq. 6.5, as the non-equilibrium scaling of the adaptive time steps depends on the relaxation time  $\tau$ . With Equation (3.10), the lattice viscosities  $\nu_n$  and  $\nu_o$  for the old and the new time step are calculated. Eq. 3.6, 3.7 and 3.8 can then be used to compute the modified local relaxation times for each cell,  $\tau_{s,n}$  and  $\tau_{s,o}$ , with  $\nu_n$  and  $\nu_o$ . Equation (6.5) must be modified to include the local

Table 6.1: Effect of the parameter change for a drop falling 2.5 times its radius. The average fill fraction Deviation  $\mathcal{E}$  is shown for each grid size, measured in comparison to a simulation without any parameter changes.

Grid Size	48	64	96	128	192	256
$\mathcal{E}$	0.000757	0.000357	0.001034	0.000542	0.000428	0.000537

relaxation time from the turbulence model. This is done by calculating the scaling factor  $s_\tau$  using the local relaxation times as

$$s_\tau = s_t(\tau_{s,n} / \tau_{s,o}). \quad (6.7)$$

Combining the turbulence model and the adaptive time steps in this way enables the simulation of high velocities without stability problems. Nevertheless, small time steps require more LB steps to compute the solution. Section 7, below, will demonstrate how to combine the techniques presented so far with an algorithm to adaptively coarsen the computational grid inside of the fluid domain with the goal of reducing the computational effort required for each LB step.

Note that the steps described so far require roughly as many computations as a collision step. Thus, the rescaling is only performed when  $|\mathbf{u}_{\max}|$  is larger than  $|\mathbf{u}_{\text{thresh}}| \cdot \xi$  or smaller than  $|\mathbf{u}_{\text{thresh}}|/\xi$ , where  $\xi$  is used to control the amount of time step changes. A value of  $\xi = 5/4$  is used here. It also needs to be prevented that the time step is decreased right after it was enlarged. This, in contrast to decreasing the time step size, is uncritical, as it is ensured that no high velocities currently appear in the simulation. A delay of four times the grid resolution in time steps yields good results here. As the time step is infrequently changed in comparison with the number of LB steps, it requires very little computational time (less than 1% for the presented simulations). The method can also be applied to flows without a free surface or without forcing. However, depending on the problem the advantages might disappear, e.g. due to constantly high velocities. Thus, the following chapter will present results for test cases where the method is applicable.

## 6.2 Validation and Performance

The correctness of the following simulations will be determined by comparing  $\mathcal{E}$ , which is the average deviation of the fluid fraction values  $\epsilon$  over all cells. The fluid fraction deviation measurement effectively compares the difference of the position of the free surface for two given configurations. If the configurations are completely different,

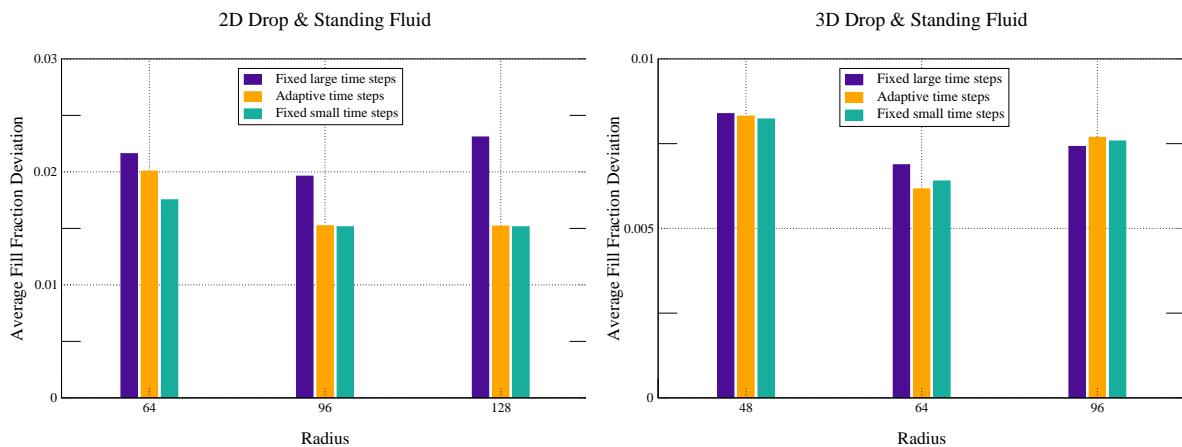


Figure 6.2: Results for the test case of Figure 6.1 in two dimensions to the left, and three dimensions to the right.

its value will be close to one, while values close to zero indicate a similar shape of the fluid. The measurements are normalized by the total number of measured points to compare simulations of different sizes, and average the measurements at different times during the course of the simulation. The fluid fraction deviation values are thus computed as

$$\mathcal{E} = \frac{1}{t_{\text{total}}} \frac{1}{n_{\text{total}}} \sum_{t=1}^{t_{\text{total}}} \sum_{\mathbf{x} \in \Omega} |\epsilon_{\text{ref}}(\mathbf{x}, t) - \epsilon(\mathbf{x}, t)|, \quad (6.8)$$

where  $\epsilon_{\text{ref}}$  are the fluid fraction values of the corresponding fine-resolution reference simulation,  $\Omega$  is the size of the domain ranging from 0 to 1 in each spatial dimension, and  $t_{\text{total}}$  is the number of timesteps to average over. Likewise,  $n_{\text{total}}$  is the total number of chosen points where  $\mathcal{E}$  is measured at. For example Table 6.1 was generated using points in 64 intervals along each axis, thus using  $n_{\text{total}} = 4096$  points in total.

A first two-dimensional test to validate our rescaling procedure is a falling drop of radius 0.1 (the size of the computational domain being 1.0) falling for 2.5 times its radius. During the course of this movement the parameterization is changed 6 times (for a resolution of  $48^2$ ) to 9 times (for a resolution of  $256^2$ ). These simulations use a  $\mathbf{u}_{\text{thresh}} = 0.05$ , which causes more parameterization changes than really necessary, and were compared to a simulation using the same resolution with a fixed parameterization. In Table 6.1 the results of these experiments are shown. It can be seen that the values of  $\mathcal{E}$  are independent of the actual resolution. Furthermore, the shape of the drop is not disturbed by the rescaling, which is evident from the very small deviations, with usually  $\mathcal{E} < 0.001$ .

A more realistic test case is shown in Figure 6.1. Here a drop with radius 0.1 is falling into a resting fluid of height 0.25. The results for experiments with various grid sizes in 2D can be seen on the left in Figure 6.2. Here each group of three fill fraction deviation values shows a comparison of different time step parametrizations. One of the three simulations used for comparison is run with a large time step (left bars), the next with adaptive time stepping (middle bars), while the last simulation (right bars) uses the smallest time step used in the corresponding simulation with adaptive time steps. The  $\mathcal{E}$  values for each grid size shown in the graph were computed by a refer-



Figure 6.3: A simulation of a falling drop with a grid resolution of  $240^3$  requiring less than 6800 steps, in contrast to more than 16000 that would have been necessary to run the simulation with the smallest time step.

ence simulation with twice this resolution parameterized to run with a small time step. Thus the left- and rightmost bars for each test case use parameterizations according to the largest and smallest time step sizes used during the course of the simulation for the middle bars. The simulations with lattice resolution  $64^2$  were initially parameterized with  $\omega = 1.87$ ,  $\mathbf{g} = (0, -5.34 \cdot 10^{-4})$  running for approximately 1000 steps. For the larger grid resolutions the parameters were scaled to keep the Reynolds number constant (hence for the test cases with a  $128^2$  resolution  $\omega = 1.678$ ). As the graph shows, the parameterization changes for time step and Mach number adaption yield the expected results lying close to the both extremes of parameterization.

The right graph in Figure 6.2 shows results for running the previous test case extended to 3D. The only difference here is that due to memory limitations the reference simulation uses 1.5 times the shown grid resolution, instead of the factor 2 for the 2D cases. Again the results with adaptive time steps lie in the expected range. Note that these test cases were chosen to ensure that even the initial large time step size remains stable. The method presented here also allows the stabilization of simulations where the occurring velocities cannot be determined from the start. In these cases the parameterization will be automatically changed in order to stabilize the simulation. The method can then save significant amounts of computational time by reducing the number of necessary LB steps. As an example, the simulation shown in Figure 6.3 requires 6757 LB steps with adaptive parameterization while running the simulation with the smallest time step size would have required 16200 steps. Thus, using the method presented here, this test case runs 2.4 times faster than without it. While not all simulations will benefit this much, they will not run slower using the adaptive parameterization, as the computational cost for checking whether to perform a parameterization change or not is negligible in comparison to the cost of each step.



# Chapter 7

## Adaptive Grids

This section will first give an overview of the standard approach to LB simulations on multiple grids. Afterwards the adaptive coarsening algorithm will be explained – its goal is to efficiently compute the motion of the free surface. Thus, the criterion for coarsening is given by the distance to the free surface. In Section 7.3, the accuracy of the method will be validated with an error metric that measures the difference of two free surface positions. Afterwards performance measurements for two different simulation setups with varying grid resolutions will be presented.

### 7.1 Grid Refinement

In [FH98], Filippova et. al developed an algorithm to couple LB simulations of different resolutions. The coupling of the different grids is done by setting boundary conditions for adjacent grids in transfer cells. This transfer of information between the grids requires a rescaling of the DFs similar to Eq. 6.5. In addition, the values have to be interpolated in space and time for the transfer from coarse to fine grids. This approach is usually used to refine a simulation grid around regions of interest, to save computational time by using a fine grid in this region only, or alternatively to increase the accuracy of the computation by refining the grid in important regions. This method has e.g. been used in [YMLS03] to compute simulations of an airfoil on a grid with refined blocks. M. Rohde recently proposed an alternative approach for grid refinement

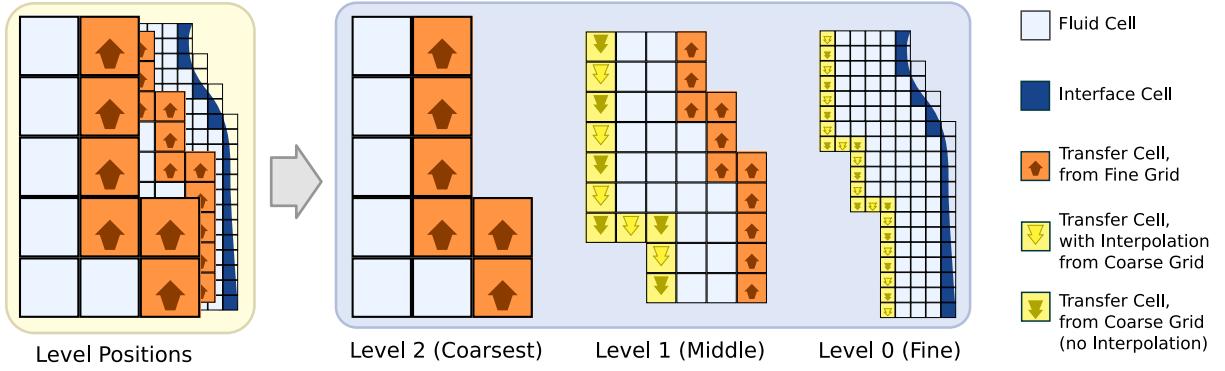


Figure 7.1: This picture shows an example of a coarsened fluid region near the free surface with 2 levels of coarser grids. To the left the transfer cell layers for coupling adjacent levels can be seen.

with the LBM, see [Roh04] for details. However, as this method requires an additional filtering step to ensure stability, this work is based on the algorithm described in [FH98].

Figure 7.1 illustrates how the transfer between a fine and a coarse grid is realized. In the following,  $c$  and  $f$  subscripts will denote variables on the coarse and fine grids, respectively. Hence, the DF  $f_{c,i}$  is a coarse grid distribution function for the direction of the velocity vector  $e_i$ , with  $f_{f,i}$  being its counterpart on the fine grid. As can be seen in Figure 7.1, the grid spacing  $\Delta x_c$  and  $\Delta t_c$  on the coarse grid are twice those of the fine grid. According to Eq. 3.11 this means that the relaxation time needs to be calculated with the corresponding parameters for each grid. Reformulating Eq. 3.11

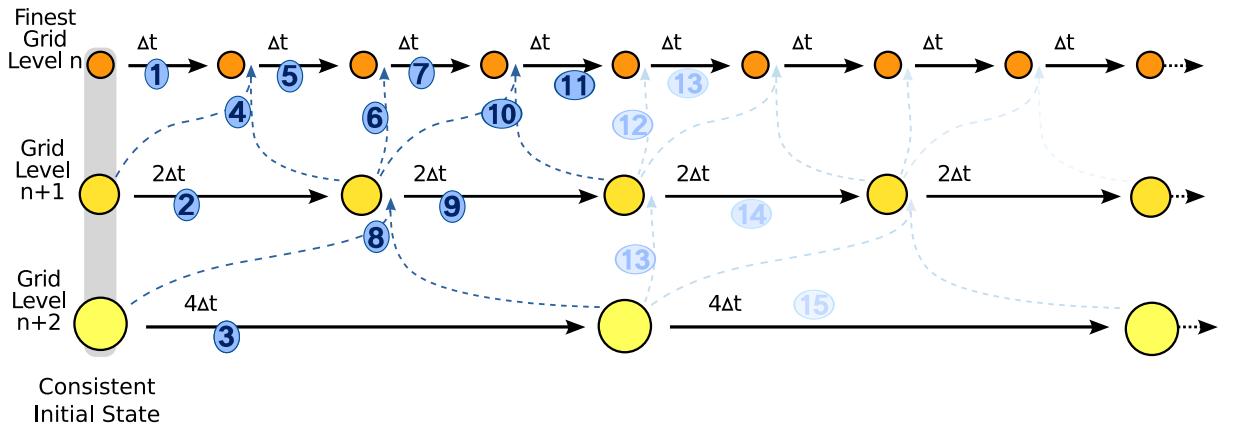


Figure 7.2: Here the effect of the different time step sizes for multiple simulation grids is shown. The numbers indicate the order in which the steps are performed. Dashed arrows indicate interpolation, while straight arrows from one circle to another represent LB steps with the indicated time step length.

and 3.10 using  $\Delta x_c = 2\Delta x_f$ , the relaxation time for the coarse grid is calculated by

$$\tau_c = \frac{1}{2}(\tau_f - \frac{1}{2}) + \frac{1}{2}. \quad (7.1)$$

In Figure 7.1 two kinds of transfer cells are shown: one for transfer from fine to the coarse grid, and vice versa. Due to the arrangement of the grids, the fine grid cells lie at the same position as the coarse grid nodes, thus data for a cell of the coarse grid transfer cells is taken directly from the corresponding fine grid cell. As the macroscopic properties such as pressure and velocity of the fluid are the same on both grids, these are not changed during the transfer. However, due to the different relaxation times, the non-equilibrium parts of the DFs have to be rescaled with

$$f_{c,i} = f_{f,i}^{eq} + s_{cf} [f_{f,i} - f_{f,i}^{eq}], \quad (7.2)$$

where  $s_{cf}$  is calculated as

$$s_{cf} = \frac{2(\tau_c - 1)}{(\tau_f - 1)}. \quad (7.3)$$

Note that Eq. 7.3 has a singularity for  $\tau_f = 1$ . This is, however, unproblematic for turbulent flows, as the low viscosity will always result in values of  $\tau$  close to 1/2. For a transfer in the other direction, from the coarse to the fine grid, Eq. 7.2 becomes

$$f_{f,i} = f_{c,i}^{eq} + \frac{1}{s_{cf}} [f_{c,i} - f_{c,i}^{eq}]. \quad (7.4)$$

Likewise, yellow fine grid transfer cells (marked with an filled downward arrow) again lie at the same positions as coarse grid cells, thus their DFs are transferred directly with Eq. 7.2. However, especially in three dimensions, most of the fine grid transfer cells are those marked with an outlined downward arrow. For these, the information from the coarse grid has to be interpolated spatially. Hence, instead of the values  $f_{c,i}$  and  $f_{c,i}^{eq}$  of Eq. 7.2, the DFs of the coarse grid are first interpolated to compute the corresponding values at the position of the fine grid cell. As described e.g. in [YMLS03], a second order interpolation is usually performed spatially.

In addition to saving operations by reducing the total number of computational cells, the number of time steps to be performed on coarser grids is reduced, as each time step on a coarse grid is twice as large as that of the next finer grid. Thus for two fine grid LB steps, only a single one has to be performed on the coarse grid. This, however, means that for one of the two fine grid steps, the grid transfer also has to include temporal interpolation of first or second order. An overview of the basic time step scheme for a total of three coupled grids is given in Figure 7.2.

## 7.2 Adaptive Coarsening Algorithm

In this thesis the view is taken that the simulation is defined by a global uniform fine grid that can be augmented with auxiliary coarser grids to accelerate the computation – at the price of a possibly reduced accuracy. This *adaptive coarsening* will be described in the next paragraphs. It will be shown how to adaptively perform a coarsening of

the fine grid simulation using a set of cell flag based rules, and how to ensure stability of the transfer between the different grid levels.

For dynamic problems, such as free surface flows or flows with moving obstacles, the techniques described in Section 7.1 cannot be applied without modifications. In [CKTR03] and [Kra01] an algorithm based on the work of Filippova et. al is used to increase the accuracy of a simulation by adaptively refining the grid around an obstacle or a bubble in the fluid. As this work is focused on the simulation of free surface flows the region of interest, that needs to be accurately computed, is the free surface itself. Hence, the simulation of this surface is performed on a fine computational grid, while the accuracy of the computation inside of the fluid may be less important. In the following an approach to adaptively coarsen the grid inside of large fluid regions by dynamically changing a set of coarser grids according to the movement of the surface on the fine grid will be described. The criterion for coarsening is thus given by the distance of a cell to the free surface. An alternative would be to allow also the coarsening of smooth free surface regions with few details. However, this would cause problems for the mass conservation with the mass flux given by Eq. 4.3 and make generating a triangulated surface more complicated.

It is thus ensured that all interface cells are treated on the finest grid. Likewise, obstacle boundaries are calculated on the finest grid. Similar to the notation used in multi-grid literature [TOS01], the fine to coarse grid transfer will be denoted with *restriction*, while the coarse to fine grid transfer will be denoted with *prolongation*.

## Boundary Cell Conversion

To adapt the coarse grids to the movement of the free surface, while keeping the transfer cell layers consistent, a set of rules to determine when to refine or coarsen a grid region was developed. The handling of the adaptive coarsening requires five passes in total, each of which, however, only applies to a single type of transfer cell. For these flag checks access to its neighborhood flags and its neighborhood flags on the next finer grid are necessary. The first three passes handle refining the coarse fluid regions, e.g. when the free surface comes near the coarsened grid region, while passes four and five handle coarsening fluid regions where the free surfaces has moved away from. It would be possible to perform some computations of the passes in parallel, but they only take a small part of the overall computational time, as will be explained in more detail in Section 7.4. Hence, in accordance to the implementation, each pass will be explained as a separate sweep over the cell flags. In the following, the five cell types shown in Figure 7.3 will be distinguished: fluid, unused, from-fine, from-coarse and to-fine.

The following rules are applied to all coarse levels. For the first level of coarsening, it is ensured that the coarsened region keeps a distance of one cell layer to the free surface, while subsequent coarsened levels ensure that they keep a distance to the restriction region of the next finer level. The following explanation will therefore focus on *from-fine* and *to-fine* cells, which are equivalent to interface cells for the finest coarse level. Due to the alignment of grids as described in Section 7.1 the fine grid neighbor  $c_f$  of a coarse grid cell  $c_c$  at position  $(i, j, k)$  is obtained by accessing cell  $(2i, 2j, 2k)$  on the fine level.

- Fluid: these are valid fluid cells treated as described in Section 2. They are not interpolated or used for interpolation.
- Unused: these cells are not included in the simulation similar to the empty cells that represent regions without fluid.
- From-Fine: DFs for these cells are transferred from the adjacent fine grid.
- From-Coarse: Likewise, DFs are transferred from the next coarser grid (possibly with interpolation).
- To-Fine: the DFs of these cells are used to interpolate the from-coarse transfer cells on the finer level. During the simulation they are treated as normal fluid cells.

Figure 7.3: The five different cell types required for grid transfer.

**Pass 1:** During the first pass, *from-fine* transfer cells on the coarse grid are checked for consistency. They are removed if the fine grid cell is not used for interpolation to a finer grid itself. Thus, if  $c_f$  is a *from-fine* or *to-fine* cell,  $c_c$  is converted to an unused cell. In this case fluid cells in the neighborhood of  $c_c$  have to be converted into *from-fine* cells, to ensure a closed transfer cell layer.

**Pass 2:** The second pass checks whether there are any unnecessary *from-coarse* cells. It only affects the coarse grid layer. One of these cells can be converted to a fluid cell when there are no unused cells in its neighborhood. Hence, the transfer cell is not required in the prolongation region. Likewise, a *from-coarse* cell can be turned to unused, if none of its neighbors are fluid cells. A special case for *from-coarse* cells is necessary to prevent a double transfer between grids. It is not desirable to have two *from-coarse* cells at the same position on different grids. Thus for a *from-coarse* cell  $c_c$ , it has to be checked whether  $c_f$  is a *from-coarse* cell as well. If this is the case,  $c_c$  has to be converted into a fluid cell, reinitializing its neighborhood to keep a closed layer of *from-coarse* cells.

**Pass 3:** After this, from-fine cells are checked for conversion to fluid cells. This has to be done when the corresponding fine grid cell is a *from-coarse* cell, meaning that the

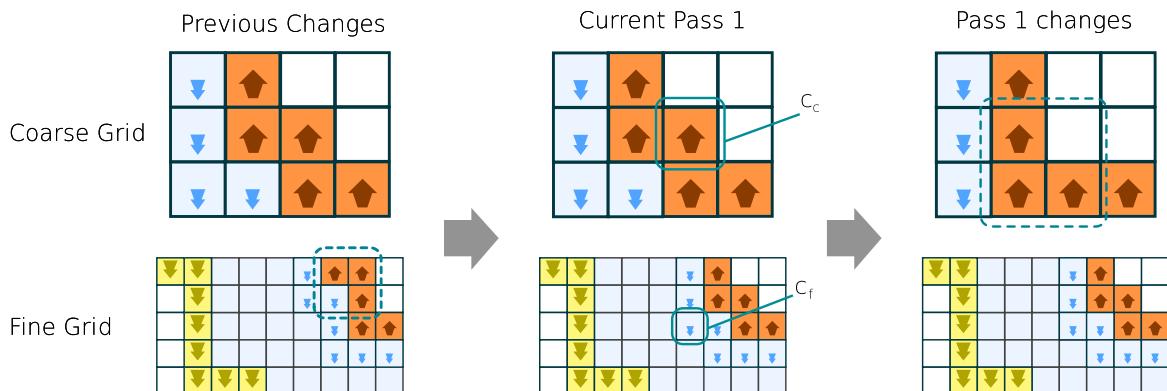


Figure 7.4: Pass 1.

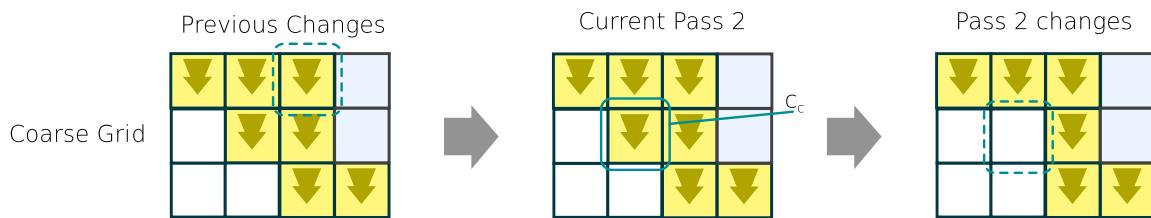


Figure 7.5: Pass 2.

finer grid transfer layer has moved away from the prolongation transfer layer on the coarse grid. In consequence, the *from-coarse* transfer cell layer of the finer grid has to be updated, turning *from-coarse* and fluid cells in the fluid region of the coarser grid into unused cells, and adding new *from-coarse* cells at the moved border.

These three passes are enough to ensure a refinement of coarsened regions when there is an inward movement of the free surface and the prolongation regions. The next two passes are similarly used to handle moving the restriction regions outwards, once the free surface moves away from it. Thus, passes four and five handle coarsening the computational grid.

**Pass 4:** For the coarsening it is first necessary to check whether an empty cell is a candidate for a *from-fine* transfer. This is the case if its fine grid neighbor is a valid fluid cell, and not a *from-fine* or *to-fine* cell. The empty cell is then turned into a *from-fine* transfer cell and initialized by a transfer of the DFs from the fine grid.

**Pass 5:** The last pass thus checks whether a *from-fine* cell can be converted into a fluid cell, coarsening the region around it. This is possible when all fine grid neighbors are valid fluid cells, not *from-fine* or *to-fine* transfer cells. Furthermore, the neighborhood of the *from-fine* cell on the coarse level must not contain any unused cells. If these criteria are met, the *from-fine* cell is turned into a fluid cell. Due to the previous checks, its neighborhood is already valid. Afterwards, all fine grid cells lying between the coarse grid cell and its neighbors have to be checked to reinitialize the *from-coarse* transfer cell layer. Fine grid cells in the center of eight valid fluid coarse grid cells are

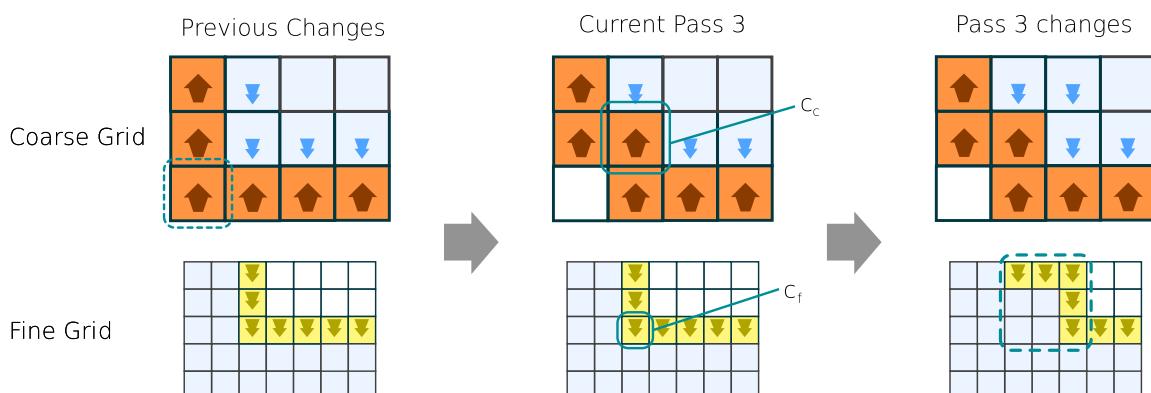


Figure 7.6: Pass 3.

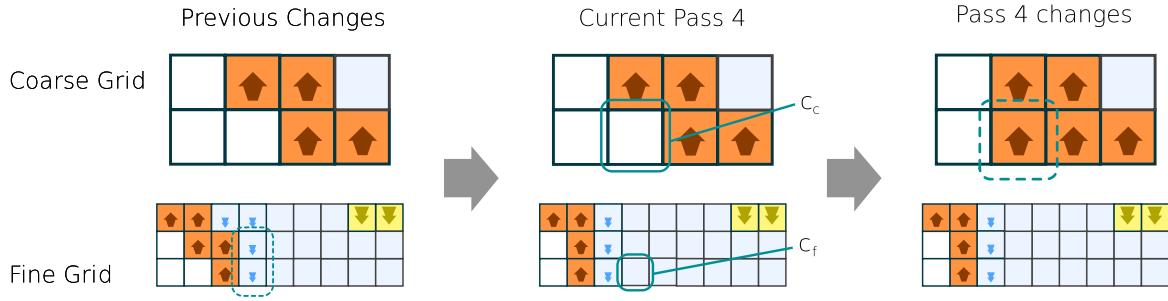


Figure 7.7: Pass 4.

directly turned into unused cells. Fine grid cells lying between fluid cells on the coarse grid have to be converted to *from-coarse* cells, while remaining *from-coarse* cells without fluid neighbors are removed from the simulation by setting them to unused.

Although the cell conversion does require 5 passes in total, the neighborhood checks are confined to small regions as linear instead of second-order spatial interpolation is applied for the prolongation. This is essential for the simplicity and efficiency of the conversion rules, as irregularities of the coarse grid transfer layer for the free surface would otherwise require checks in large neighborhoods of the *from-coarse* transfer cells. In Section 7.3 it will be shown that the accuracy of the linear interpolation is computationally sufficient by comparing it directly to a second order interpolation.

## Grid Transfer

These conversion rules are checked before each coarse grid step. They are enough to ensure a valid and closed layer for both restriction and prolongation. As direct transfers across multiple grid levels are prevented, and the restriction transfer layer of the first coarsened level does not cover interface cells, the resulting simulation regions usually span 2-3 fluid cells between their transfer layers. After adapting the grid, restriction and prolongation are performed to set correct boundary conditions for the actual LB step.

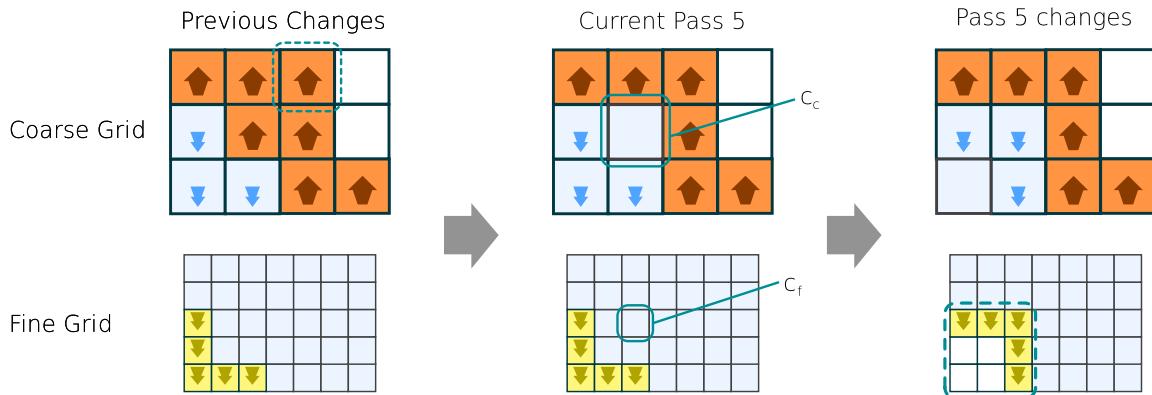


Figure 7.8: Pass 5.

The transfer of DFs on the boundaries is done by including the modified relaxation time of the turbulence model in Eq. 7.2. After interpolation of the DFs, the modified relaxation times  $\tau_{sc}$  and  $\tau_{sf}$  are calculated with Eq. 7.3 using the viscosities  $\nu_c$  and  $\nu_f$ , respectively. Finally, the scaling factor  $s_{cf}$  is calculated with

$$s_{cf} = \frac{2(\tau_{sc} - 1)}{(\tau_{sf} - 1)}, \quad (7.5)$$

and used instead of Eq. 7.3 for the transfer scaling factor in Eq. 7.2 and Eq. 7.4.

A remaining problem of the algorithm discussed so far is, that simulations with low viscosities are disturbed by artifacts that are caused by the overlapping grids. An example of this problem can be seen in Figure 7.9. The artifacts are caused by pressure fluctuations near obstacles and become noticeable as self-reinforcing patterns at the grid boundaries that cause strong disturbances of the flow field. The problem here is that according to the description of Section 7.1 the restriction is done using a single fine grid cell, analogous to *injection* in a multi-grid algorithm. The resulting information is used on the coarse grid, and during the subsequent steps propagated to the fine grid again two cells further in the fluid region at the *from-fine* transfer cells. To break up this pattern of information flow, a restriction that takes into account all fine grid cells within the fine grid neighborhood of a coarse grid cell is used. This is shown on the right side of Figure 7.9 for a two-dimensional example. Thus, the cells that were previously not taken into account for the restriction also contribute to the coarse grid transfer cells. For interpolation a simple gauss kernel gives good results. Thus, the interpolated DFs  $\tilde{f}_{f,i}$  to use with Eq. 7.4 are calculated as

$$\tilde{f}_{f,i}(\mathbf{x}) = \sum_{\alpha=1}^{19} f_{f,i}(\mathbf{x} + \mathbf{e}_\alpha) \frac{w_\alpha}{w_{\text{total}}} \quad (7.6)$$

with

$$w_\alpha = e^{-|\mathbf{e}_\alpha|} - e^{-2 \cdot 3}, \quad w_{\text{total}} = \sum_{\alpha=1}^{19} w_\alpha \quad (7.7)$$

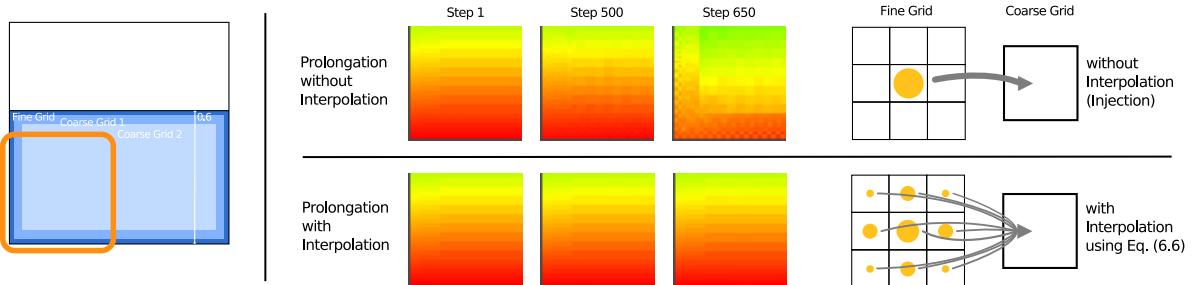


Figure 7.9: Example of artifacts that occur for a simple standing fluid test case with a resolution of  $128^2$  and two coarse levels. Each picture shows the density distribution in the lower left corner of the fluid, where green values indicate  $\rho = 1.0$  while a red color indicates larger values. The upper row of pictures was created without any interpolation for the prolongation, while the lower row makes use of Eq. 7.6.

This interpolation requires more accesses to fine grid DFs for restriction, but effectively prevents the development of the artifacts described above.

In conclusion, the algorithm proceeds with the following steps for all levels that are advanced at a given time:

1. Start with coarsest grid level.
2. Adapt the grid:
  - (a) perform refinement passes 1,2 and 3,
  - (b) perform coarsening passes 4,5.
3. Set the boundary conditions with restriction and prolongation.
4. Perform the LB step (for the finest level this includes handling the free surface).
5. Continue with the next finer grid.

The accuracy of both the interpolation scheme and the adaptive coarsening algorithm will be evaluated in the next section.

### 7.3 Validation

The accuracy of the different grid transfer methods will be determined as explained in Section 6.2 by measuring the fill fraction deviation  $\mathcal{E}$  from a reference simulation averaged over time and a certain number of measurement points. In Figure 7.11 the grid resolution of the reference simulation was used to set the number of measurement points. Note that  $\mathcal{E}$ , in contrast to error metrics from the multi grid literature, does not

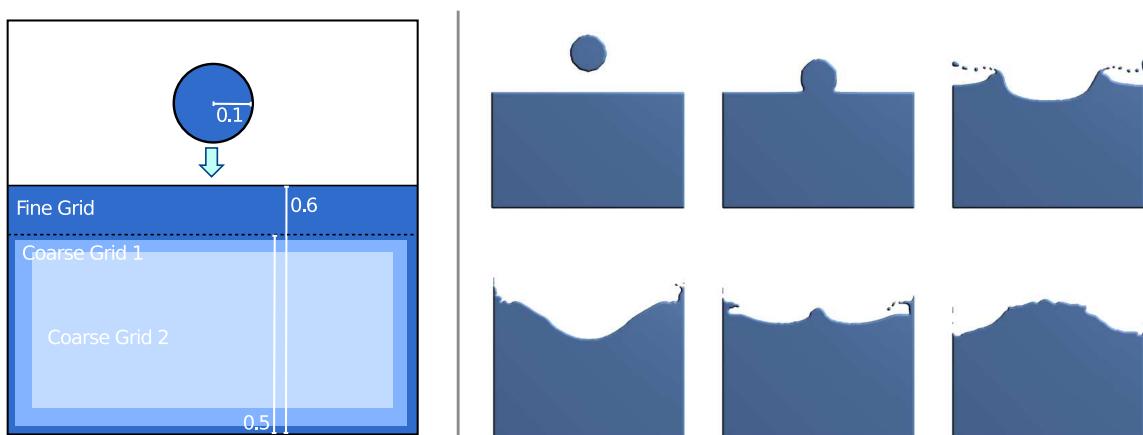


Figure 7.10: Falling drop test case setup for interpolation accuracy measurements with static coarsening.

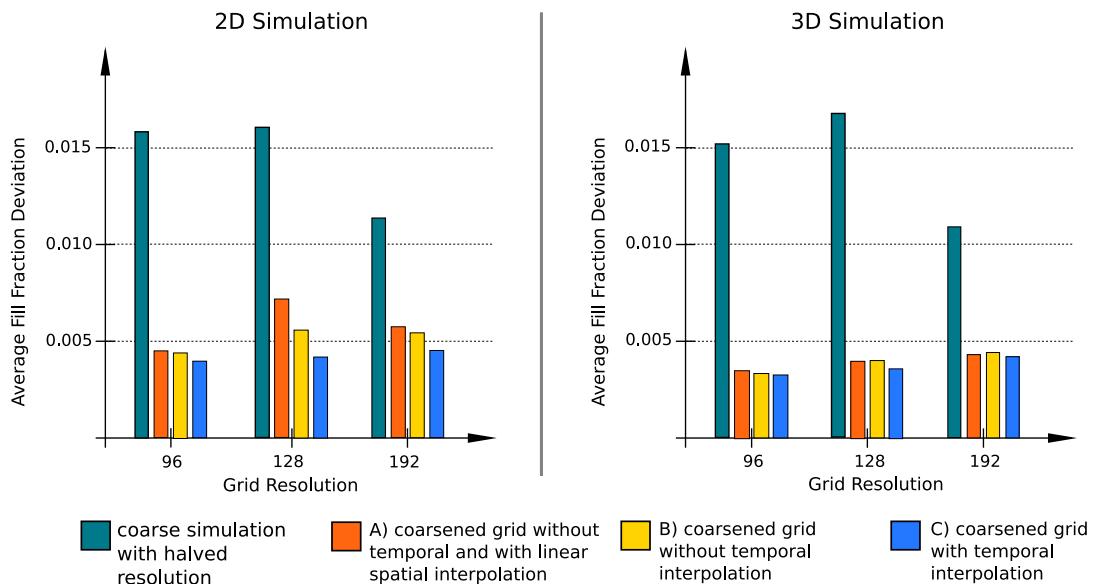


Figure 7.11: Accuracy measurement for the interpolation test case of Figure 7.10.

measure the error caused by representing the problem on a coarser grid, but only the position of the free surface.

The following test cases were parametrized to represent a cubic domain of 0.1m length with water and earth gravity. Hence,  $\nu' = 1^{-6}$  [m<sup>2</sup>/s] and an acceleration of  $\mathbf{g}' = (0, -9.81, 0)^T$  [m/s<sup>2</sup>] are chosen.

## Static test case

The different interpolation methods will be tested using the simulation setup described on the left side of Figure 7.10. In Figure 7.10 the domain with a side length of 1.0 is filled 60% with fluid. A drop placed at height 0.8 with radius 0.1 is accelerated by a gravitational force to fall into it. Several pictures from a two-dimensional simulation with a resolution of 192<sup>2</sup> can be seen on the right side of Figure 7.10. In three dimensions, the drop is converted to a cylinder with rounded edges, to match the two-dimensional test case. As indicated in Figure 7.10 the test cases use a fixed coarsening of the lower half of the domain, to reduce any influences of dynamic changes of the coarsened region during the course of the simulation. The two coarse grid regions are visible as lighter areas. The free surface comes close to the coarsened regions, but the setup was chosen to keep it at a distance that ensures correct calculations with a static coarsening.

Figure 7.11 shows results in two and three dimensions, to the left and right, respectively, each for three grid resolutions. The reference simulation is a simulation run on an uncoarsened grid with the shown resolution. The coarsened simulation is run three times with the following interpolation methods:

- A) without temporal interpolation and with linear spatial interpolation,
- B) without temporal interpolation and with second order spatial interpolation,
- C) with linear temporal and second order spatial interpolation.

Each of these runs was performed with two levels of coarsening – one with halved, and the coarsest one with  $1/4$  of the original resolution. For reference, the simulation is also run once on a grid with half the shown resolution (referenced as *coarse* in the following) and without any additional coarsened grids.

Throughout the runs it can be seen that the adaptively coarsened simulations are significantly more accurate than the corresponding runs with halved resolution. Furthermore, there is only a slight difference between the different interpolation variants. The interpolation method C is the most accurate one, as was expected. The other two, however, only show small decreases in accuracy. This can be attributed to the fact that for the coarsened grids the free surface and the obstacles are still calculated on the finest grid everywhere. These regions determine the overall motion of the fluid. Thus, in contrast to test cases such as [YMS02], the coupling with the coarser grids is sufficiently accurate without the temporal interpolation, and more importantly, without second order spatial interpolation. Former allows us to use the grid compression technique [PKW<sup>+</sup>03] on all grids, as only a single time step needs to be stored in memory. It also saves one third of the total memory accesses that are required to interpolate the coarse grid DFs to the fine grid, as for each second interpolation step the temporal interpolation would require access of two DFs instead of one. The linear spatial interpolation greatly simplifies the handling of the grid adaptivity, and significantly reduces the number of memory accesses. For linear interpolation, fine grid cells that lie between 2, 4 and 8 coarse grid cells require the same number of DF accesses for each interpolated one. With second order spatial interpolation, it would, however, require 4, 16 and 64 DF accesses, respectively. For the test case described above with a grid resolution of  $128^3$  this means that on average only 130773 DFs have to be accessed and interpolated for method A, instead of 363253 for interpolation method B.

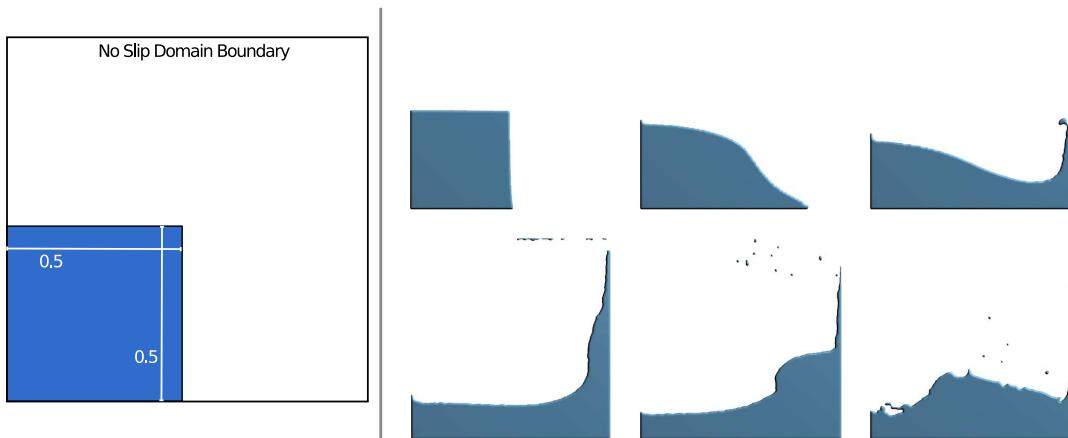


Figure 7.12: Breaking dam test case setup for accuracy measurements for the adaptive coarsening procedure.

## Dynamic test case

To validate the accuracy of the adaptive coarsening technique described in Section 7.2 the breaking dam setup described in Figure 7.12 was used. The domain is filled with a region of fluid in the lower left corner, taking up a quarter of the domain volume. The gravity causes the fluid to splash against the upper right corner, and form a wave travelling back towards the left wall. This fluid movement obviously requires constant updates of the coarsened region.

Accuracy measurements of  $\mathcal{E}$  computed with Eq. 6.8 are shown in Figure 7.13. Here again a coarse simulation with half the shown resolution and an adaptively coarsened simulation (using interpolation method A) are compared to a simulation run on a homogeneously fine grid. It can be seen that the accuracy of the adaptive simulations is slightly less than the accuracy of the static test cases with coarser grids. However, throughout the runs they are more accurate than the coarse simulation, while requiring significantly less LB cells than the fine simulation. Still, the free surface is represented with the same amount of details as the fine simulation. The next section will show performance results of simulation runs to illustrate the speedup that is possible using the adaptive coarsening method.

## 7.4 Performance

Before analyzing the overall performance, it is important to know how the workload is distributed between the different parts of the algorithm. Thus, a run of the test case shown in Figure 7.14 was profiled. It was parametrized with a resolution of  $256^3$  and run on a single Opteron CPU for 2500 LB steps. The adaptive coarsening algorithm was used for 3 coarse levels in addition to the finest one.

As can be seen in Table 7.1, the majority of the computations are necessary for

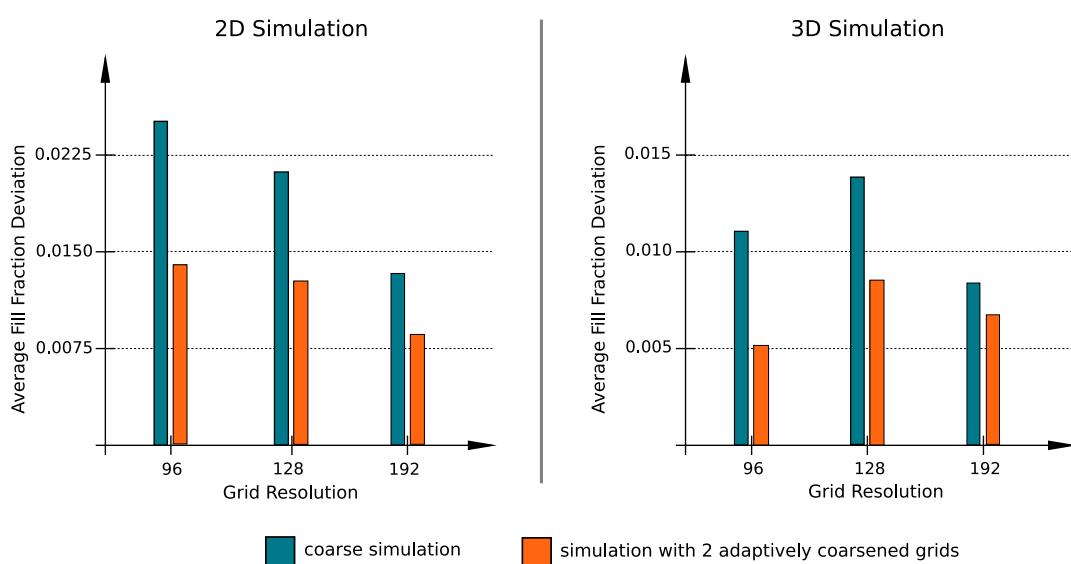


Figure 7.13: Accuracy measurement for the adaptive coarsening test case of Figure 7.12.

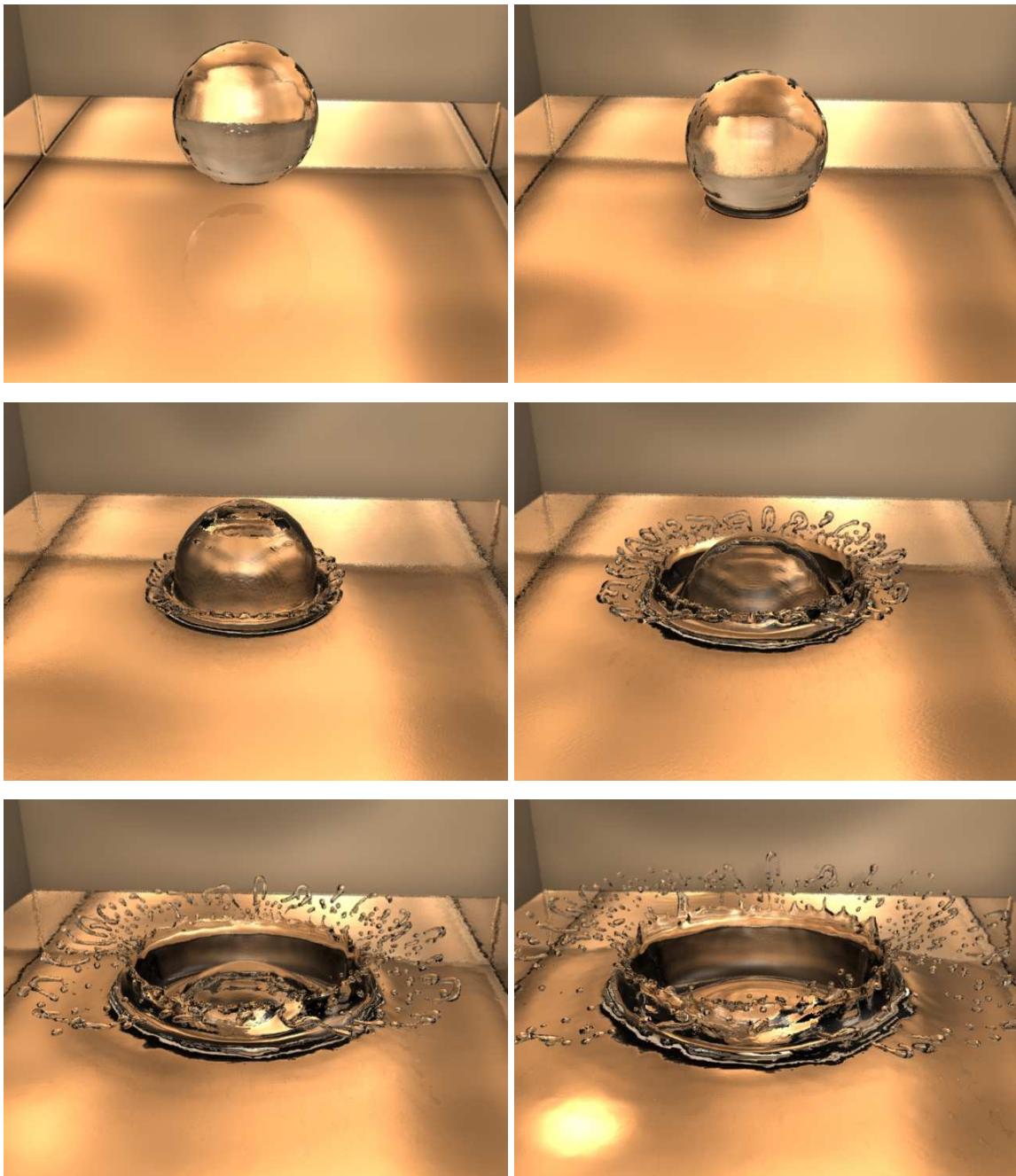


Figure 7.14: Images of the falling drop simulation with a grid resolution of  $480^3$  and the adaptive coarsening algorithm.

advancing the finest grid and computing the free surface boundary conditions. The adaptive coarsening itself requires more computational effort than the LB steps on the coarse grids themselves. This is due to the fact that the coarse grids usually only contain relatively few fluid cells, and the adaptive coarsening includes the calculation of the grid transfer, which for a single cell requires computations similar to a normal LB cell update.

Usually, the performance of LB programs is measured with the number of cell updates per second: *MLSUPS* (million lattice site updates per second). However, this is

Table 7.1: Workload distribution for an exemplary simulation.

Procedure	Workload percentage
Fine grid LB steps	73.46%
Adaptive coarsening	14.27%
LB steps of all coarse grids	7.25%
Other code	5.02%

not valid anymore once coarsened grids are involved due to the differing cell sizes and constantly changing number of overall cells. In this case, it is crucial how much faster the overall simulation is done with the adaptive grids, in comparison to a standard simulation using a single grid level. The following tables shows several MLSUPS measurements only to illustrate the performance of the implementation without adaptive coarsening for a falling drop test case as shown in Figure 7.14.

Table 7.2 shows that the basic LB implementation yields a high performance on a variety of CPU architectures. For the OpenMP version the Intel compiler was used<sup>A</sup>, while the other serial measurements were performed with GCC<sup>B</sup> compilations. A high performance of the serial version is important, as a poor implementation might yield larger speedups when combined with the adaptive coarsening technique – even when the overall performance is bad. Note that other highly optimized codes, such as presented in [Igl03, Don04], achieve MLSUPS rates more than twice as high as those given in Table 7.2. However, these codes usually only consider the basic LBM with bounce back boundary conditions. The MLSUPS measurements from Table 7.2 on the other hand include several extensions that influence the absolute performance: the free sur-

<sup>A</sup>Intel Compiler Version 8.1, which produced the fastest code on this machine in comparison to other versions which were available.

<sup>B</sup>Gnu Compiler Collection, Version 4.0.3.

Table 7.2: Performance measurements of the basic free surface simulation code without adaptive coarsening on different architectures with up to four processors. Measurements from the OpenMP version of the solver are marked as such, the other measurements are taken from optimized serial versions.

CPU/Architecture	MLSUPS
Opteron, 2.2 GHz (248), OpenMP	1.44
Opteron, 2.2 GHz (248)	1.82
Pentium4, 3.2 GHz (Northwood)	1.84
Athlon64, 2.4 GHz	2.04
Dual-Opteron, 2 · 2.2 GHz (248), OpenMP	2.87
Dual-Itanium2, 2 · 1.5 GHz (Madison), OpenMP	2.91
Dual-Core2, 2 · 2.4 GHz (Conroe)	4.21
4-way Opteron, 4 · 2.2 GHz (848), OpenMP	5.40

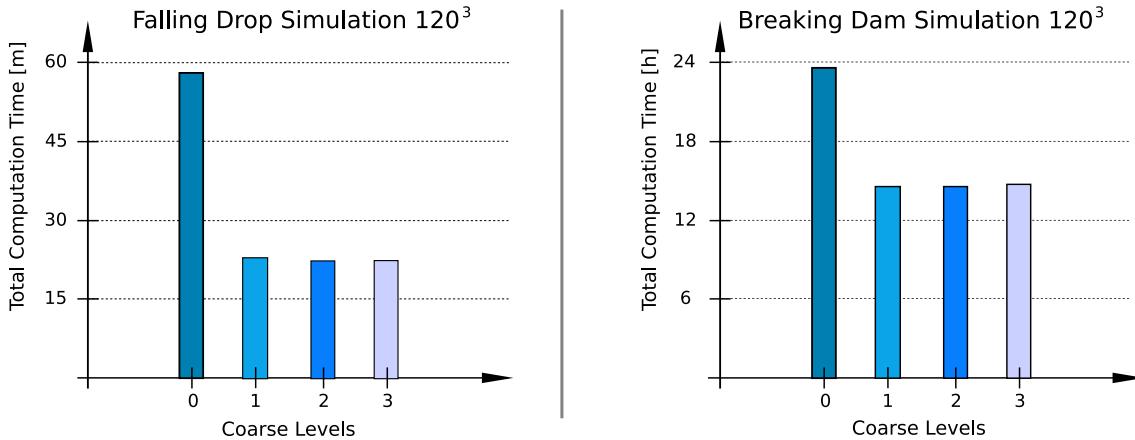


Figure 7.15: Performance for a resolution of  $120^3$  on a single Pentium4 CPU with 3.2GHz.

face treatment, the turbulence model, and support for different boundary conditions.

Two test cases will be used to demonstrate the achievable speed-ups:

Case A (Figure 7.14): a falling drop test case, and

Case B (Figure 7.18): a breaking dam problem.

Both cases were run in three different sizes:  $120^3$ ,  $240^3$  and  $480^3$ . Each graph shows the total computation time with a different number of coarse grids. The simulation of the first bar to the left is run only on the finest level, while the others use up to three levels of adaptive coarsening.

In Figure 7.15 the performance for the relatively small resolution of  $120^3$  on a Pentium4 CPU with 3.2GHz is visible. For test case A a speed up of ca. 2.5 is achieved once the first coarsened level is used. Due to the small size of the domain, additional levels of coarsening do not yield a further speedup. Similarly for test case B, the first coarsened level yields a speedup of ca. 1.6. The lower speedup in comparison to test case A can be attributed to the fact that test case B has a smaller volume of fluid and exhibits a larger number of thin fluid sheets. Hence, it is a harder problem for the adaptive coarsening technique.

Figure 7.16 shows performance results for a larger domain resolution of  $240^3$  on a dual Opteron node. Each CPU has 2.2GHz in this case, and OpenMP was used to parallelize the algorithm for the shared-memory architecture. As was demonstrated above, the majority of the work is done on the finest grid – thus the parallelization is only applied to the traversal of the finest grid level. The parallelization of the algorithm will be discussed in more detail in Section 8. In contrast to the  $120^3$  runs, more than a single level of coarsening yields a further speedup for test case A with a resolution of  $240^3$ . In total, a speedup of 4.14 and 2.75 is achieved for test case A and B, respectively.

The last performance results of Figure 7.17 are for a resolution of  $480^3$  on a four way Opteron node (again with 2.2 GHz for each of the four CPUs). As before, the traversal of the finest grid was parallelized with OpenMP. For a simulation without adaptive coarsening, test case A now requires more than 54 million cells. The total speedup with 3 coarsened grids is 3.85 in this case, and 3.16 for test case B.

In conclusion, the adaptive coarsening can be used to speed up simulations with large bodies of fluid without significantly reducing the accuracy of the computations. As this combination of free surface model, turbulence model, adaptive time steps, moving obstacle boundary conditions and adaptive grids is the final state of the underlying free surface solver of this thesis, the next section will discuss how to increase the overall performance by making use of multiple computing cores.

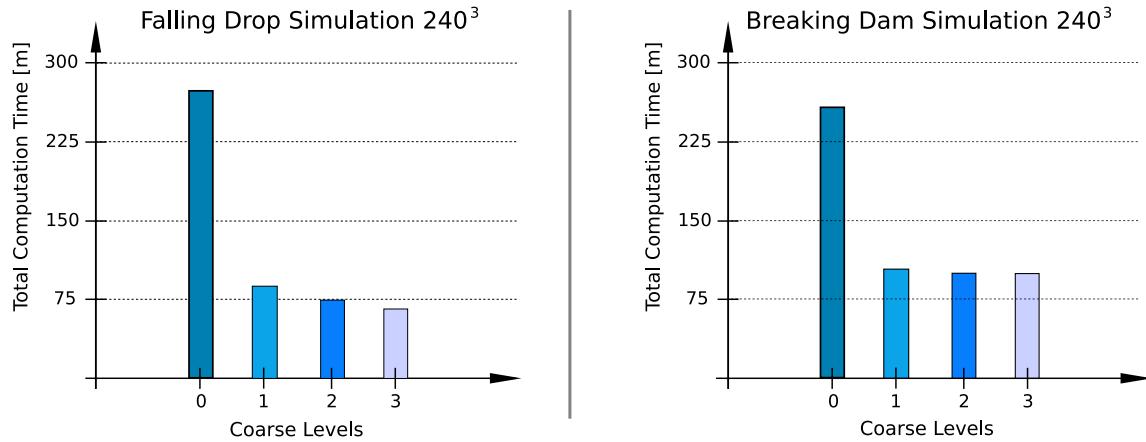


Figure 7.16: Performance with OpenMP parallelization for a resolution of  $240^3$  on a dual Opteron Node (each CPU with 2.2GHz).

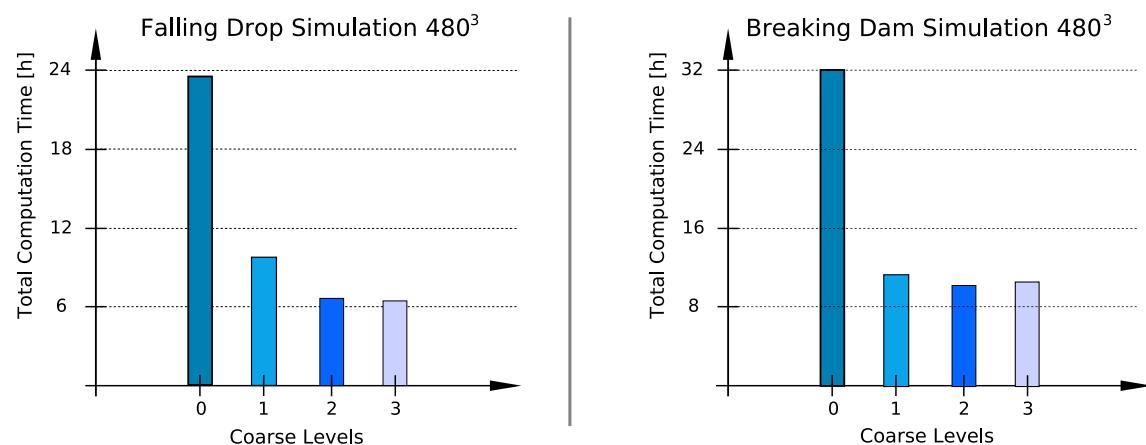


Figure 7.17: Performance with OpenMP parallelization for a resolution of  $480^3$  on a four way Opteron Node (each CPU with 2.2GHz).

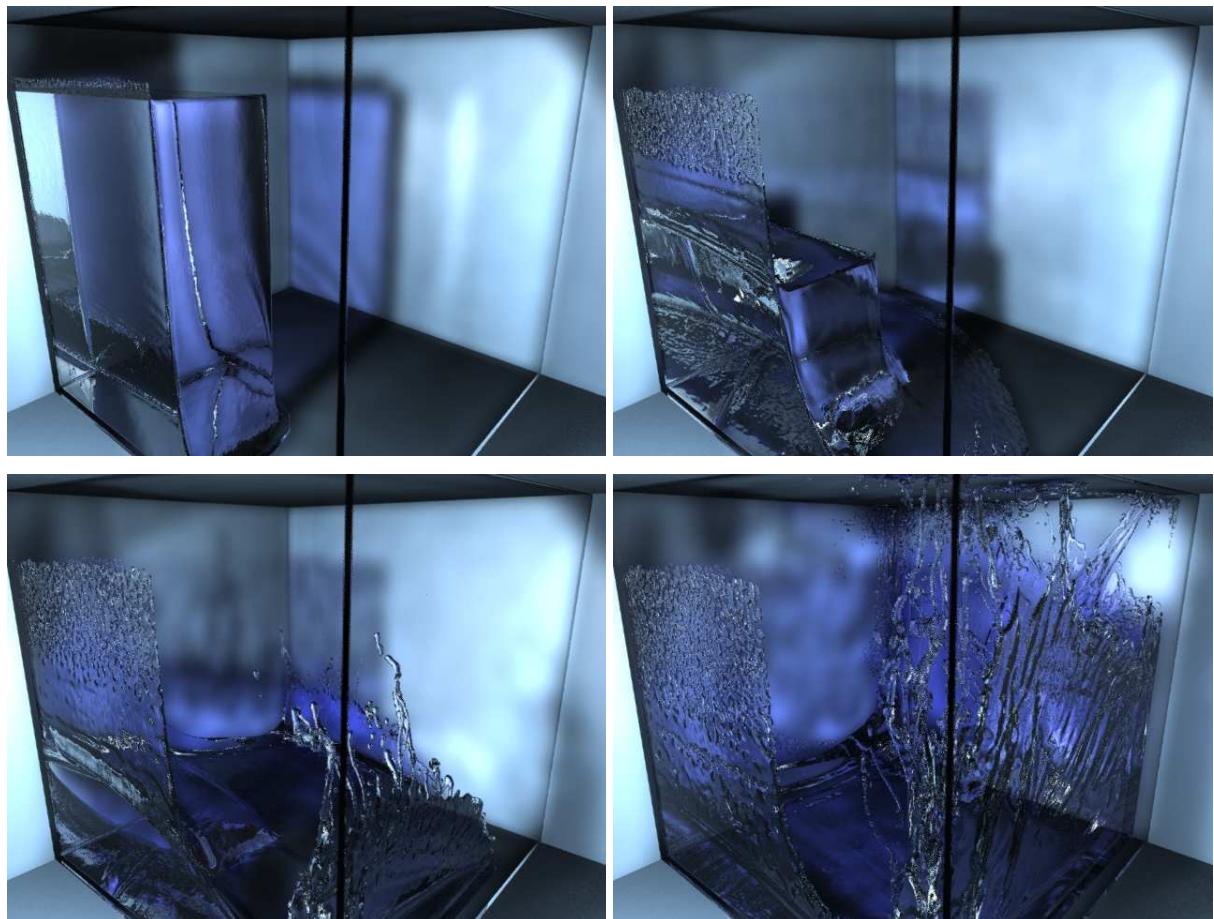


Figure 7.18: Pictures of the breaking dam test case, again with a grid resolution of  $480^3$ .

---



# Chapter 8

## Parallelization

The following section will present an algorithm to perform free surface simulations on machines with shared and distributed memory architectures. Performance results for different test cases and architectures will be given. The algorithm for parallelization yields a high performance, and can be combined with the adaptive time steps and grids of the previous sections. On the other hand, it is not fully mass conserving, and as such is targeted towards the creation of animations. For parallel algorithms that guarantee mass conservation, and possibly require load balancing, other approaches, such as [Poh07], or variants of [BT99] could be used.

### 8.1 OpenMP Parallelization

*OpenMP* is a programming model for parallel shared-memory architectures, and has become a commonly used standard for multi-platform development. If supported by the compiler, the developer can insert pragmas into the source code (assuming the use of the C/C++ programming language) to indicate which regions of the program should be executed in parallel. An overview of the whole API is given in, e.g., [CDK<sup>+</sup>01]. The advantage of this approach is that it is relatively easy to use, due to the high-level nature of the OpenMP instructions. The same source code can also be used to compile a single threaded application, if the compiler does not support OpenMP. On the other hand the OpenMP standard slightly limits the flexibility when the high-level

---

commands, such as automatic loop parallelization, are used. These loops, e.g., have to conform to certain rules in order to be admissible for the OpenMP instructions.

## OpenMP Implementation

As was shown in Table 7.1, the main part of the computations of the solver (ca. 73%) need to be performed for the computations of the finest grid. The parallelization thus aims to speed up this central loop over the finest grid of the solver. A natural way to do this would be to let the OpenMP compiler parallelize the outermost loop over the grid, usually the z direction. However, as for the solver of this thesis the grid compression technique [PKW<sup>+</sup>03] is used, this would violate the data dependencies for a cell update. With grid compression, the updated DFs of a cell at position  $(i, j, k)$  in the grid are written to the position  $(i-1, j-1, k-1)$ . This only works for a linear update of all cells in the grid. Instead, to use grid compression with OpenMP, the DFs of cell  $(i, j, k)$  are written back to the position  $(i, j, k-2)$ , as shown in Figure 8.1. This allows the update of all cells of an xy plane in arbitrary order. Note that this modified grid compression only requires slightly more memory than the original version (a single line of cells along the z direction, assuming the same x and y grid resolution).

Hence, the loop over the y component is parallelized, as shown in Figure 8.2. This is advantageous over shortening the loops along the x direction, as cells on a line along the x axis lie successively in memory. Long loops in this direction can thus fully exploit spatial coherence of the data, and prefetching techniques of the CPU if available. After each update of a plane the threads have to be synchronized before continuing with the next plane. This can be done with the OpenMP *barrier* instruction. Afterwards, the cells of the next xy plane can again be updated in any order. In the following a gravitational force along the z axis will be assumed. This usually causes the fluid to spread in the xy plane, which justifies a domain partitioning along the x and y axes.

Slight changes to the loop core are required to ensure the correct update of global

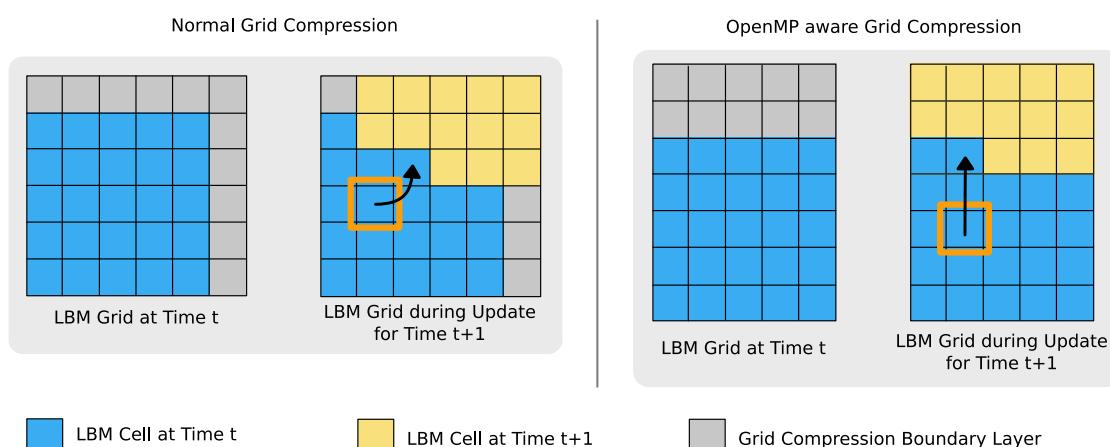


Figure 8.1: Comparison of the normal grid compression, and the grid compression for OpenMP. Instead of copying an updated cell to its diagonal neighbor, the cell is copied to a target cell in two cells distance (along the y direction for this 2D example, along the z direction for an actual 3D implementation).

values. The loop for example computes the total mass and volume of the fluid that is currently being simulated, as these values are necessary for resizing of the time step. Inflow and outflow objects can change the overall mass even though the algorithm is mass conserving, and due to the relaxed incompressibility constraint, the volume of the fluid is slightly changing during the course of the simulation. These values need to be computed separately for each thread, and are added after the parallelized region was finished. The lists of filled and emptied cells, as described in Section 4.2, also require additional treatment. Each thread creates a separate list of cells that are concatenated before the cell flag reinitialization is performed.

## OpenMP Performance Measurements

Performance measurements of the OpenMP parallelized solver can be seen in Figure 8.3. The graphs show absolute time measurements for a fixed number of LB steps, as MLSUPS measurements are not suitable for simulations with the adaptive coarsening of Section 7. The setup is a falling drop, as shown in Figure 7.14. The left graph was measured on a 2.2GHz dual Opteron workstation<sup>A</sup>, with a grid resolution of  $304^3$ . The graph to the right of Figure 8.3 was measured on a 2.2GHz quad Opteron workstation<sup>B</sup>, using a resolution of  $480^3$ . For the dual nodes, as well as the quad nodes, the CPUs are connected by HyperTransport links with a bandwidth of 6.4 GB/s. Each graph shows timing measurements for different numbers of CPUs, and with or without the use of the adaptive coarsening algorithm.

The results without the adaptive coarsening show the full effect of the paralleliza-

<sup>A</sup>CPU: 2 x AMD Opteron 248, 2.2 GHz, 1MB L2-cache; 4GB DDR333 RAM.

<sup>B</sup>CPU: 4 x AMD Opteron 848, 2.2 GHz, 1MB L2-cache; 16GB DDR333 RAM.

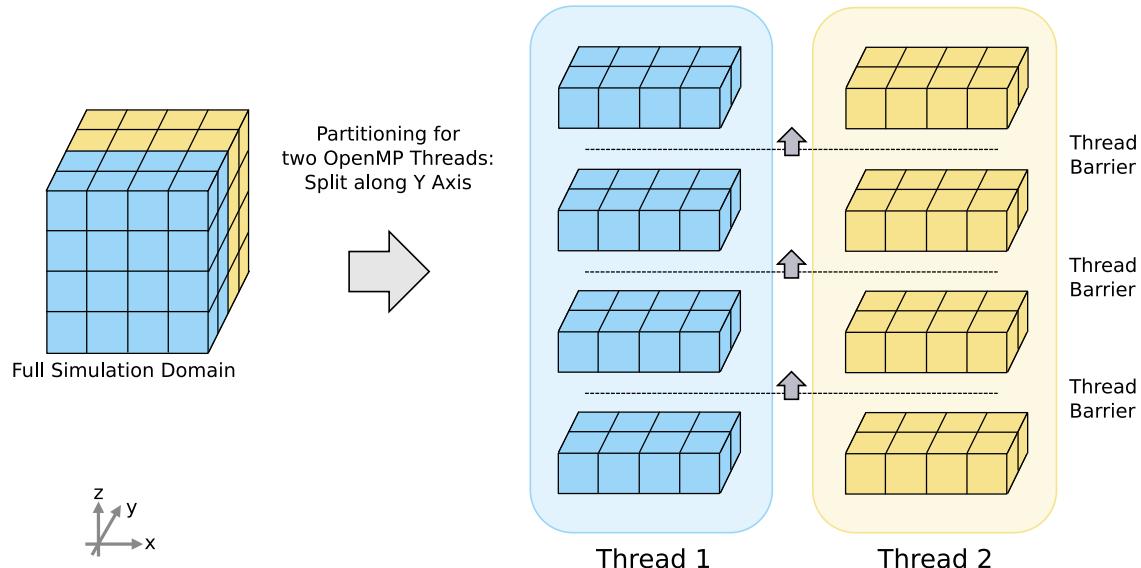


Figure 8.2: The OpenMP parallelization splits the y component of the loop over the grid.

tion, as in this case almost 100% of the computational work is performed on the finest grid. It is apparent that the speedup is directly proportional to the number of CPUs used in this case. The four blue bars of the right graph from Figure 8.3 even show a speedup of 4.11 for four CPUs. This can be explained with the architecture of the quad node – the simulation setup uses most of the available memory of the machine, but each CPU has one fourth of the memory with a fast local connection, while memory accesses to the remaining memory have to be performed using the HyperTransport interconnect. Thus, with four OpenMP threads the full memory bandwidth can be used, while a single thread, solving the same problem, frequently has to access memory from the other CPUs.

The timing results with adaptive coarsening show less evident speedups, as in this case only roughly 70% of the overall runtime are affected by the parallelization. As expected, the runtime for two CPUs is 65% of the runtime for a single CPU. The OpenMP parallelization thus yields the full speedup for the finest grid. It can be seen in the right graph of Figure 8.3 that there is a speedup factor of more than 15 between the version without adaptive coarsening running on a single CPU and the version with adaptive grids running on four CPUs. To further parallelize the adaptive coarsening algorithm, a parallelization of the grid reinitialization would be required, which is complicated due to the complex dependencies of the flag checks.

## 8.2 MPI Parallelization

For the development of applications for distributed memory machines, the *Message Passing Interface* (MPI) is the most widely used approach. In contrast to OpenMP, MPI

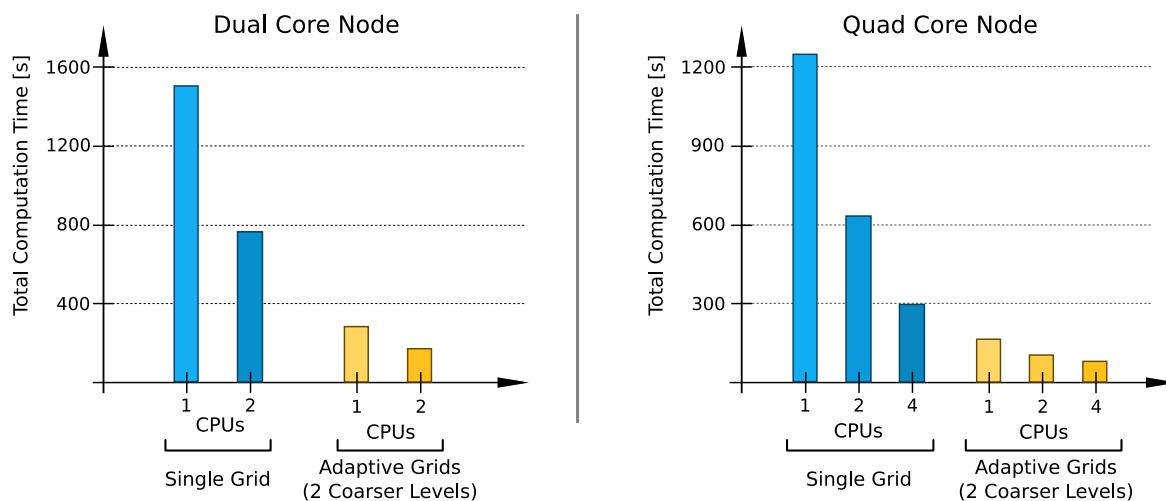


Figure 8.3: Time measurements for the OpenMP version of the solver: the runs to the left were measured on a dual Opteron workstation, while those to the right were measured on a workstation with four Opteron CPUs. The blue bars represent runs without the adaptive coarsening algorithm, while the orange bars use two coarse levels in addition to the finest one.

requires more low level work from a developer, as most of its functions only deal with the actual sending and receiving of messages over the network. Thus, a developer has to make sure that the problem to be solved is correctly split into work packages for each node in the network. MPI can then be used to exchange the information that needs to be shared among the participating nodes. Details of the introductory and more advanced functions of MPI can be found, e.g., in [GLS99] and [GLT99].

## MPI Implementation

For parallelizing the LB solver of this thesis, the domain is split along the x axis, as shown in Figure 8.4. In this figure two nodes are used, the domain is thus halved along the x axis, and a ghost layer is added at the interface of the two halves. Before each actual LB step, the boundary planes are exchanged via MPI, to assure valid boundary conditions for all nodes. As indicated in Figure 8.4, the boundary layer contains the full information from a single plane of the neighboring node. For a normal LB solver, this would be enough to perform a stream-collide-step. However, the free surface handling can require changes in the neighborhoods of filled and emptied interface cells from a previous step. All fluid cells in the layer next to the boundary layer thus have to be validated again. If one of them has an empty cell as a neighboring node, it is converted to an interface cell. This simple handling ensures a valid computation, but causes slight errors in the mass conservation, as the converted cell might have received excess mass from the former neighboring interface cell. It was found that this is unproblematic, especially for physically based animations, as the error is small enough to be negligible. For engineering applications, an additional transfer between the nodes could ensure a correct exchange of the excess mass, similar to the algorithm proposed in [PTD<sup>+</sup>04, KPR<sup>+</sup>05]. The error in mass conservation is less than 1% for 1000 LB steps.

If this scheme is used in combination with the adaptively coarsened grids, it has to be ensured that there is no coarsening of the ghost and transfer layers. Therefore, the transfer is only required for the finest grid level. A coarsening of the ghost layers would require information from a wider neighborhood of the cells, and result in the

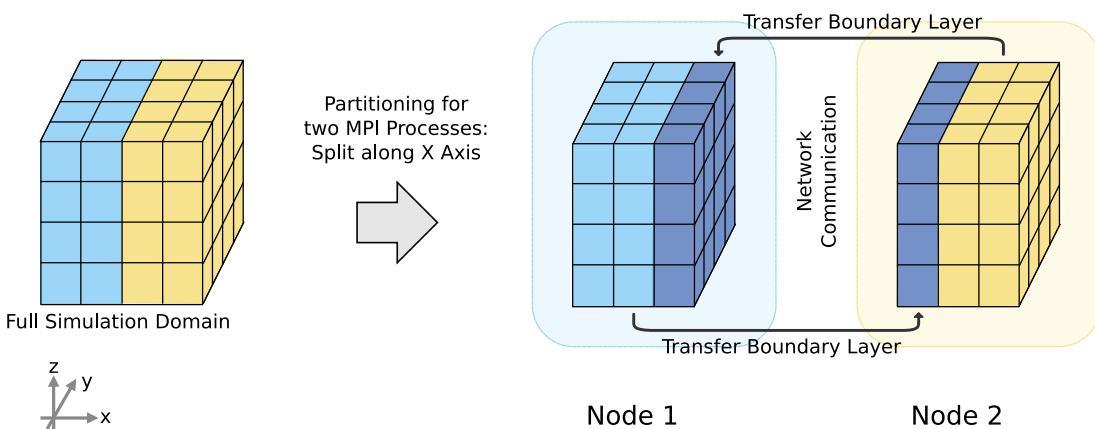


Figure 8.4: The MPI parallelization splits the x component of the loop over the grid.

exchange of several layers near the node boundary, in addition to the ghost layers of the different grid levels. As the bandwidth of the network is a bottleneck, such an increase of the exchanged data would result in a reduced performance. Hence, only the fine layers are connected with a ghost layer, and are treated similar to the free surface or obstacle boundaries to prevent coarsening in this region. As the node boundary has to be represented on the finest grid, this means that large volumes of fluid spanning across this boundary can not be fully coarsened. This parallelization scheme thus modifies the actual layout of the coarse and fine grids in comparison to the serial version of the solver.

Further care has to be taken for the adaptive resizing of the time step. This is based on the maximum velocity in the simulation, and requires the global maximum for all nodes to be computed. Due to the same allowed maximum velocity for all nodes, the maximum velocity for each MPI process is computed. The MPI function *MPI\_Allreduce* can then be used to conveniently compute the global maximum. To reduce the amount of global communication, it is sufficient to check and compute the maximum velocity in intervals. An interval of four LB steps will be used in the following.

## MPI Performance Measurements and Discussion

To measure the performance of this MPI parallelized version of the solver, the test case from Section 7 is used again (a drop falling into a basin of fluid). The timing measurements of Figure 8.5 were measured on multiple quad Opteron nodes<sup>C</sup>. Details of these quad nodes can be found in Section 8.1. The x axis of each graph shows the number of nodes used for the corresponding measurement. For each node, the OpenMP parallelization of the previous section was used to execute four OpenMP threads on each node. As the parallelization changes the adaptive coarsening, Figure 8.5 again shows timing measurements for a fixed number of LB steps, instead of MLSUPS or MFLOPS rates. The figure shows two graphs in each row: the graph to the left was measured on a grid without adaptive coarsening, while the one to the right was measured from runs solving the same problem with two levels of adaptive coarsening. The two rows show the effect of the overhead due to MPI communication: the upper row shows results for a cubic domain of  $480^3$ , denoted as test case Q in the following, while the lower row was measured with a wider channel and a resolution of  $704 \cdot 352 \cdot 352$  (test case W). The grid resolution remains constant for any number of CPUs involved (strong scaling). As the domain is equally split for the number of participating MPI nodes, test case Q results in thinner slices with a larger amount of boundary layer cells to be transferred. For 8 nodes and test case Q, this means that each node has a grid resolution of  $60 \cdot 480 \cdot 480$  with  $480^2$  boundary cells to be exchanged. Splitting the domain of test case W, on the other hand, results in slices of size  $88 \cdot 352 \cdot 352$  with  $352^2$  boundary cells to be exchanged.

Overall, the graphs without adaptive coarsening show a speedup of around 1.8 for the strong scaling. While the speedup for test case Q, from four to eight nodes, is around 1.62, it is 1.75 for test case W, due to the better ratio between computations and communication in the latter case. This effect can also be seen for the graphs with adaptive coarsening (the right column of Figure 8.5). While the curve flattens out for

---

<sup>C</sup>The nodes are connected by an InifiniBand interconnect with a bandwidth of 10Gbit/s

test case Q, there is a larger speedup for test case W. For test case Q with adaptive coarsening, the speedup factor is ca. 1.3 – 1.35, while it is between 1.45 and 1.55 for test case W. This lower speedup factor, in comparison to the test cases with only a single fine grid, is caused by the increased overhead due to MPI communication, compared to the amount of computations required for each LB step with the adaptive coarsening. Moreover, the amount of coarsening that can be performed for each slice of the grid is reduced with the increasing number of MPI processes. Practical test cases will, however, usually exhibit a behavior that is a mixture of the four test cases of Figure 8.5. An example of a large scale test case that was computed with four MPI processes, and required almost 40GB of memory, can be seen in Figure 8.7.

To evaluate the overall performance of the solver, varying rectangular grid sizes without adaptive coarsening were used to simulate problems requiring the whole memory of all participating nodes (weak scaling). While the MLSUPS rate for a single quad Opteron is 5.43 with a grid resolution of  $704 \cdot 352 \cdot 352$ , eight quad nodes with a resolution of  $1040 \cdot 520 \cdot 520$  achieve a performance of 37.3 MLSUPS, as shown in Figure 8.6. This represents a total speedup factor of 6.87 for the eight nodes.

It was demonstrated that the parallel algorithm presented here is suitable to perform efficient large scale computations. Both algorithms for OpenMP and MPI parallelization can be combined to solve large problems on hybrid shared- and distributed-memory systems. However, the algorithm does not yield the full performance when the only goal is to reduce the computational time for small problems with MPI. For large problems, the speedup will effectively depend on the setup – for large volumes

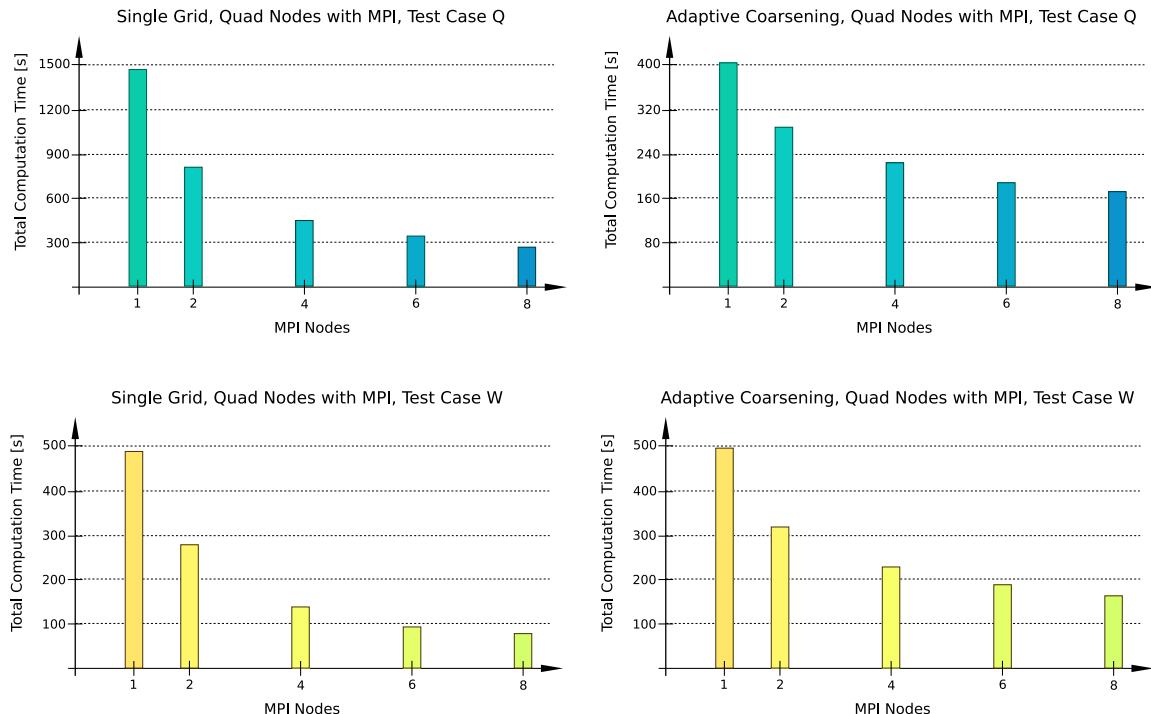


Figure 8.5: Time measurements for the MPI version of the solver running a problem with  $480^3$  (test case Q) in the upper row, and for a problem with  $704 \cdot 352 \cdot 352$  (test case W) in the lower row of graphs.

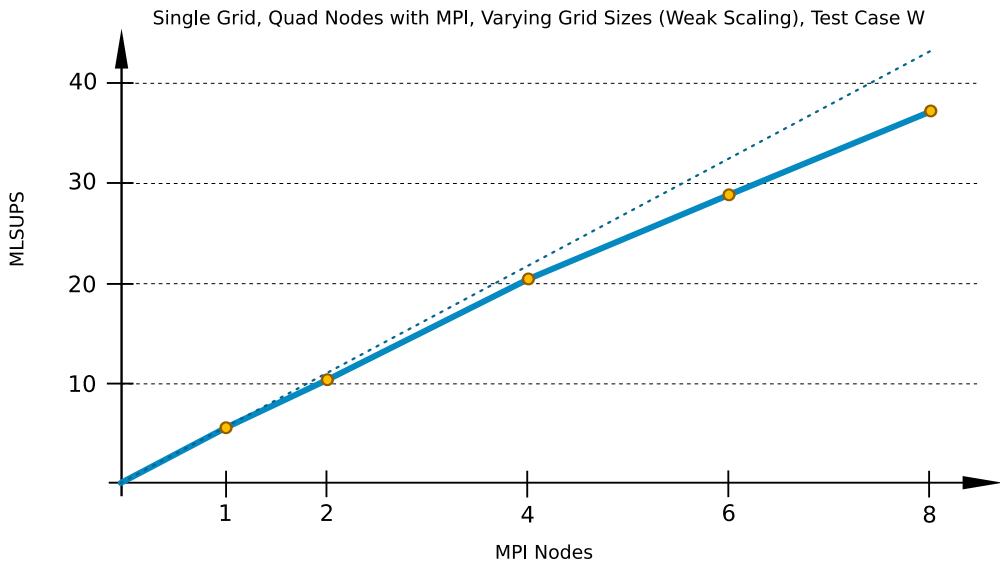


Figure 8.6: MLSUPS measurements for runs with varying grid resolutions (weak scaling) of test case W and without adaptive coarsening. The dotted line represents the projected ideal speedup according to the performance of a single quad node.

of fluid, the speedup can be around 1.3 – 1.5, while fluids with many interfaces and fine structures can almost yield the full speedup of a factor two for each doubling of the CPUs or nodes used for the computation.

For dynamic flows, an interesting topic of future research will be the inclusion of algorithms for load balancing, e.g., those described in [KPR<sup>+</sup>05]. The algorithm currently assumes an distribution of fluid in the xy plane due to a gravity along the z direction. If this is not the case, the static and equidistant domain partitioning along the x and y axes will not yield a high performance.



Figure 8.7: Pictures of a wave test case with a grid resolution of  $880 \cdot 880 \cdot 336$ . On average, only 6.5 million grid cells were simulated during each time step due to two levels of adaptive coarsening.



# Chapter 9

## Fluid Control

While realism is an important aspect for physically based animations, the practical use of fluid simulations for animation is also determined by the ability to efficiently control the behavior of the fluid. In general, animators prefer to have uncontrolled physical simulations only where absolutely necessary. Since fluid motion is typically very hard to predict, it is difficult to achieve a specific fluid behavior only by changing the corresponding global parameters. For some cases, such as animations of characters consisting of fluid, there is no real-world reference, and thus no way to really validate such a simulation. Still, it is required that such an animation looks physically plausible. Animators are mostly interested in modifying the large-scale motion of the fluid, without having to specify fine-scale detail such as small vortices or drops. High-level control is thus crucial in production environments.

The method presented in this chapter makes use of control particles, similar to [FF01]. Particles are a natural choice, as they are established tools in all major 3D-applications, can be intuitively handled, and animators are familiar with particle systems for creating various other effects. Since control particles are independent of the underlying fluid model they can be integrated easily in different flow simulation environments. In addition, many different control scenarios can be implemented, such as scripting, keyframing, or coarse-to-fine simulations. It will furthermore be shown how control particles can be automatically generated from pre-computed functions, an animated target shape, or an existing flow simulation. Directly enforcing control from the particles onto the fluid can lead to noticeable distortions of the velocity field, which is

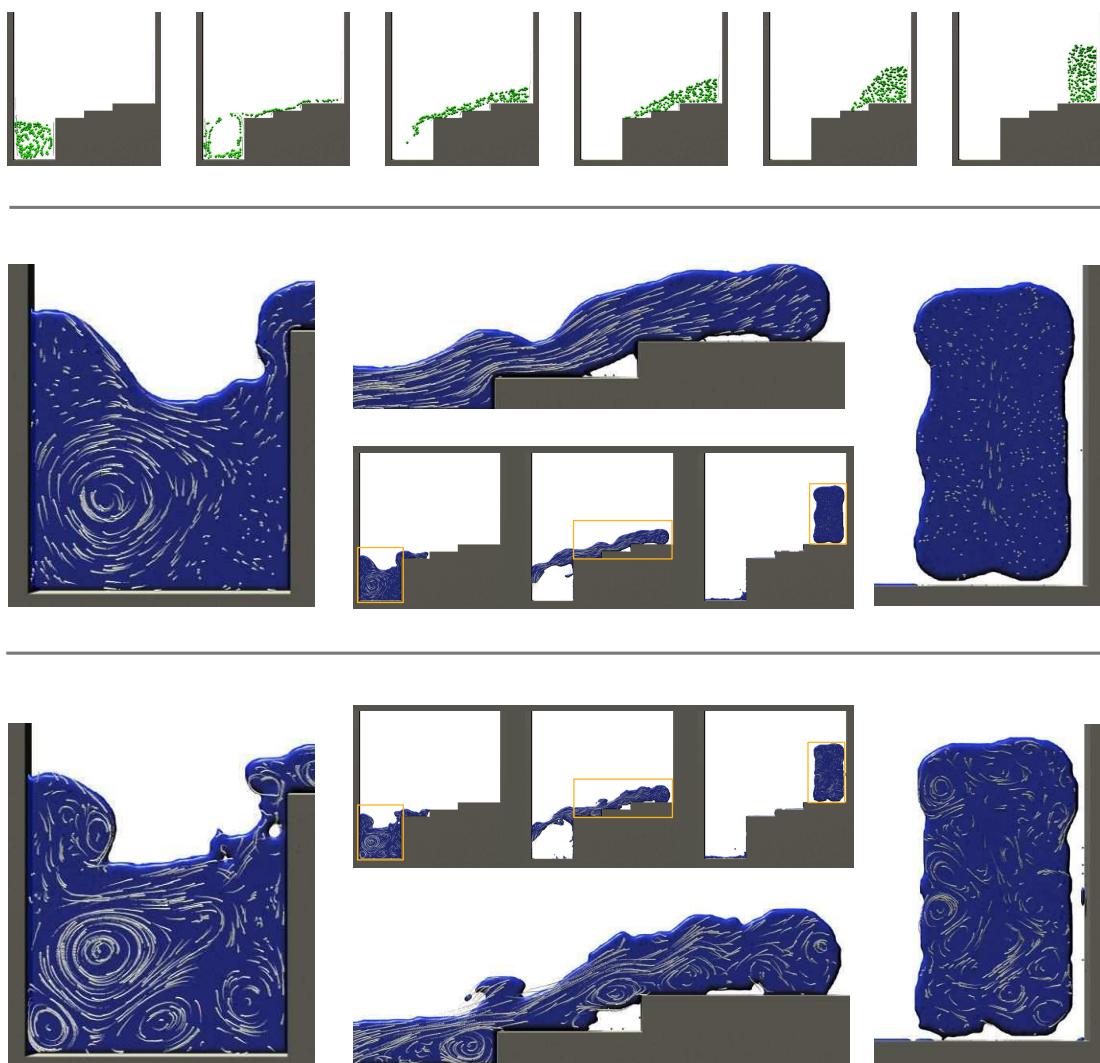


Figure 9.1: Comparison of direct velocity control (middle) and scale-separated velocity control (bottom). Both simulations are controlled by the same 250 control particles (shown in the top row) that were generated by reversing an uncontrolled simulation of fluid flowing down the stairs.

noticeable as an increased viscosity. To avoid this artificial viscosity, the velocity field is decomposed into coarse- and fine-scale components and the control forces are only applied to the low-frequency part. High-frequency components are largely unaffected, thus small-scale detail and turbulence are significantly better preserved. An example of this effect can be seen in Figure 9.1. Techniques that make use of similar decompositions are used for example in geometry processing or motion capture editing. The decomposition is achieved by smoothing the velocity field using a low-pass filter given by the influence kernels of the control particles. Velocity control forces are then computed with respect to the smoothed velocity field. Although the following description will focus on the grid based LBM, the control mechanism is also applicable to other grid

based solver (e.g., level set methods) or Lagrangian fluid solvers, such as smoothed particle hydrodynamics (SPH).

In [FM97] Foster and Metaxas were the first to propose the embedding of controllers to control pressure and velocity of the flow. This concept is further extended in [FF01] by sampling 3D parametric space curves with oriented points to locally alter the velocity of the fluid. Space curves are also used in [LF02] to model flames. These curves evolve according to physics-based, procedural, and manually defined wind fields. [FOA03] already demonstrates the capabilities of particle based fluid control for animating explosions. Rasmussen et al. [REN<sup>+</sup>04] introduce viscosity, velocity divergence and level set particles for melting, expansion and contraction of the liquid. [TMPS03] presents an optimization technique to solve for control parameters such that simulated smoke matches the given density and velocity keyframes. The efficiency is improved by adopting the adjoint method for solving the nonlinear optimization problem in [MTPS04]. As control with the adjoint method is a more general framework, it can also be applied to, e.g., particle systems and cloth [WMT06]. The authors of [PCS04], on the other hand, demonstrate an approach that makes use of radial basis functions to control flow simulations. Fattal and Lischinski [FL04] proposed the idea of driving smoke towards target smoke density states by introducing a force term and counteract diffusion of smoke by adding a gathering term to the Euler equations. This simple technique is significantly faster than the previously mentioned approaches. Hong and Kim [HK04] derive potential fields from the initial distribution of smoke and a target shape. The force field is then defined as the gradient of this potential field. Shi and Yu control both smoke [SY05a] and liquids [SY05b] by matching the level set surface of the fluid with static or moving target shapes. Velocity constraints at the boundary force the fluid into the desired shape. While for smoke a compressible fluid model can be used [SY05a], the velocity field needs to be divergence-free for liquids to guarantee mass preservation, as described in [SY05b]. To achieve this, the bounday forces are modified by solving a minimization problem that yields a divergence free velocity field.

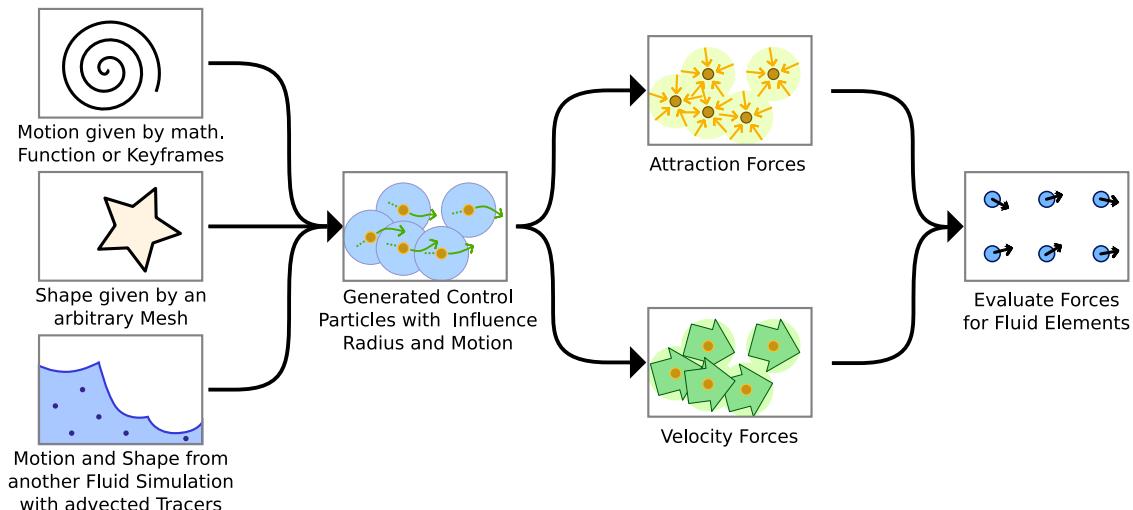


Figure 9.2: Here an overview of the particle based control framework can be seen.

The control framework uses a set of particles that locally exert forces on the fluid. Control particles are generated using an implicit function, a sequence of target shapes, or another fluid simulation. They directly induce forces to attract the fluid or influence the velocity field. It will be demonstrated that these two forces can be used for a wide variety of effects. It will furthermore be shown how the small-scale detail can be retained by applying the control forces only on the coarse flow of the fluid. An overview of the framework can be seen in Figure 9.2.

## 9.1 Generating Control Particles

In the following, three different methods to generate control particles will be described. The easiest way to create control particles is with a given pre-computed function as described in [FM97, FF01]. To perform fluid simulations with a given target shape, the initial triangle shape mesh is first regularly sampled. The control particles are then displaced for each mesh of the animation sequence using mean value mesh coordinates, as described in [JSW05]. Control particles can also be generated from other, possibly coarser, fluid simulations. Within an LBM simulation, massless tracer particles can be tracked in the fluid velocity field. Their positions are then used as control particles in a second simulation pass. Such a control simulation can usually be very coarse, and may even run in realtime to give instant feedback to an animator. It can likewise be controlled to yield the desired result. Control particle sets generated by a fluid simulation can be used to easily control large volumes of fluid. Furthermore, by changing or reversing the timing of the control particles interesting effects can be achieved. Both Figure 9.1 and Figure 9.7 make use of this approach.

## 9.2 Control Forces

A control particle  $p_i$  is given by its position  $\mathbf{p}_i$ , velocity  $\mathbf{v}_i$ , and influence radius  $h_i$ . A constant radius  $h$  is chosen as 2.5 times the average sample distance of the control particles.

Fluid attraction is controlled using a force that pulls fluid towards the control particles. In order to preserve as much of the natural fluid behavior as possible, this force is scaled down when the influence region of the control particle is already sufficiently covered with fluid. Let  $V_e$  denote the volume of a fluid element  $e$ , such as a filled grid cell for the LBM. A scale factor is defined for the attraction force as

$$\alpha_i = 1 - \min \left( 1, \sum_e V_e W(d_{i,e}, h) \right), \quad (9.1)$$

where  $d_{i,e} = \|\mathbf{p}_i - \mathbf{x}_e\|$  is the distance between  $p_i$  and the center  $\mathbf{x}_e$  of fluid element  $e$ , and  $W$  is the control particle kernel function. A linear falloff function of width  $h/2$  can be sufficient for  $W$ :

$$W(d, h) = \begin{cases} 1 & : d \leq h/2 \\ 2 - \frac{2d}{h} & : d > h/2, d < h \\ 0 & : d \geq h \end{cases} \quad (9.2)$$

However, in the following a normalized spline kernel with support  $h$  [MCG03b] will be used

$$W(d, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - d^2)^3 & : d < h, \\ 0 & : d \geq h. \end{cases} \quad (9.3)$$

Summing up the attraction forces exerted by control particles  $p_i$  on a fluid element  $e$  then yields

$$\mathbf{f}_a(e) = w_a \sum_i \alpha_i \frac{\mathbf{p}_i - \mathbf{x}_e}{\|\mathbf{p}_i - \mathbf{x}_e\|} W(d_{i,e}, h), \quad (9.4)$$

where  $w_a$  is a global constant that defines the strength of the attraction force. If  $w_a$  is negative, Equation (9.4) will result in a repulsive force.

While the attraction force pushes fluid towards control particles that are not covered with fluid yet, a second force is used to modify the velocity of the fluid according to the flow determined by the control particles. A velocity force per volume  $\mathbf{f}_v$  for the fluid element  $e$  is defined similar to the attraction force

$$\mathbf{f}_v(e) = w_v \sum_i [\mathbf{v}_i - \mathbf{v}(e)] W(d_{i,e}, h), \quad (9.5)$$

where  $\mathbf{v}(e)$  is the velocity of the fluid element  $e$ , and  $w_v$  a constant that defines the influence of the velocity force. Finally, the new total force per volume  $\mathbf{f}(e)$  acting on the fluid element is given by the sum of attraction, velocity and fluid forces

$$\mathbf{f}(e) = \mathbf{f}_a(e) + \mathbf{f}_v(e) + \mathbf{f}_f(e). \quad (9.6)$$

Here  $\mathbf{f}_f(e)$  is the force given by the physical fluid simulation, e.g., gravity. Integrating  $\mathbf{f}(e)$  gives the new velocity  $\mathbf{v}'(e)$  of a fluid element that is then used for the calculation of the equilibrium distribution functions for the LBM. In order to apply the method to a level set based solver it will be necessary to ensure that the velocity field is divergence free, e.g., as in [SY05b].

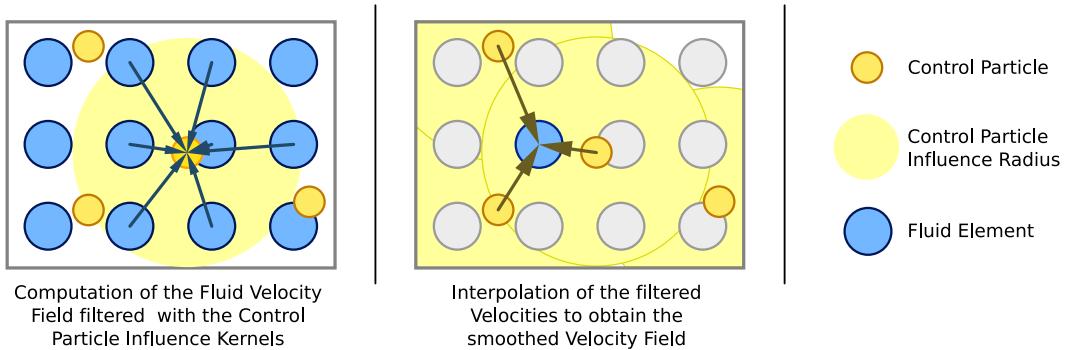


Figure 9.3: This figure illustrates the process of computing the filtered fluid velocity field for each control particle, and its interpolation back to the fluid elements.

### 9.3 Detail-Preserving Control

The velocity force of Equation (9.5) effectively leads to an averaging of the fluid velocities with the control velocities, which introduces undesirable artificial viscosity. This effect is illustrated in Figure 9.1, where 250 control particles force the fluid to flow up the stairs. The middle row of pictures is calculated with control forces as described above. Within the influence of the control particles, the fluid is unable to develop small-scale vortices and turbulent behavior.

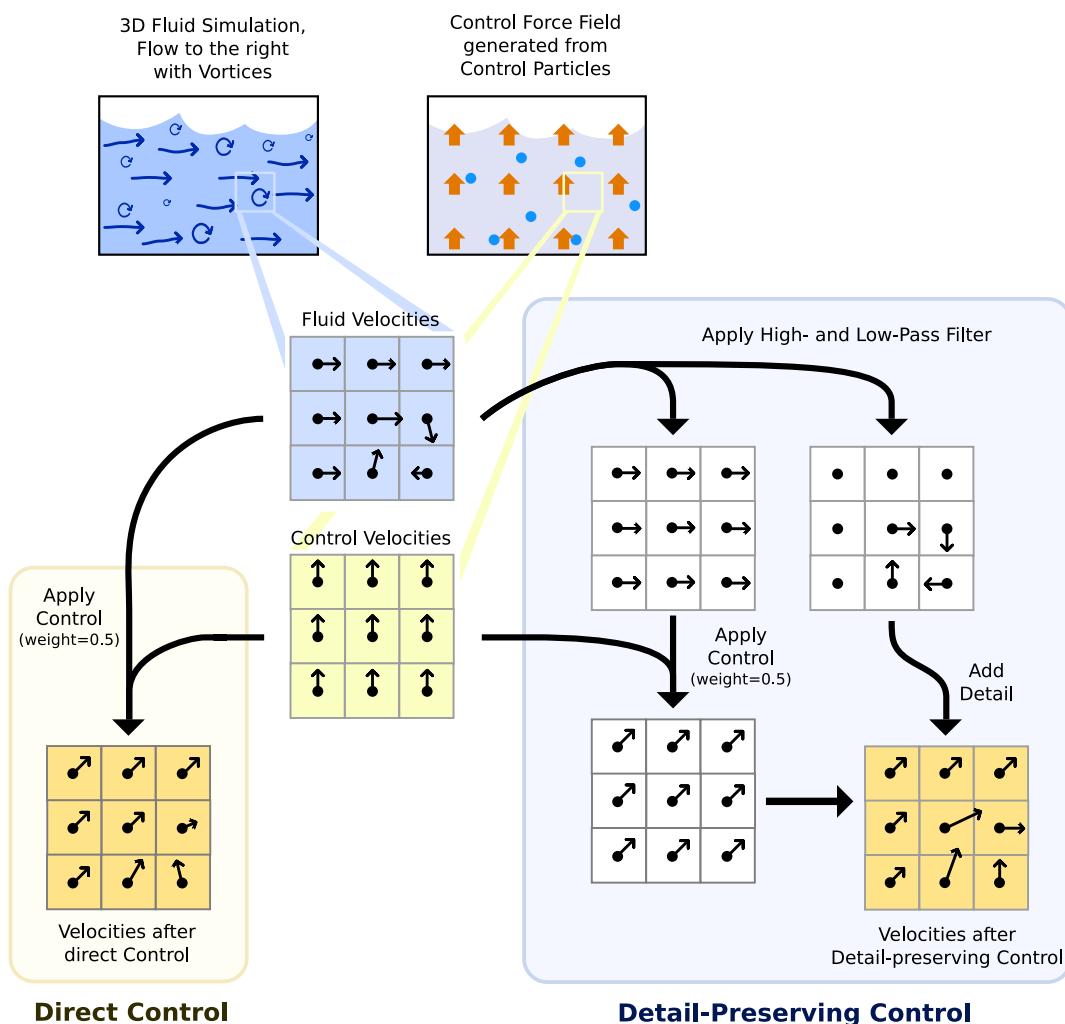


Figure 9.4: Overview of the two possibilities for applying velocity control - the lower left velocity field shows the effect of direct velocity control. The velocities of the vortex are distorted significantly. The lower right velocity field shows the preserved vortex with the improved velocity control method.

It needs to be ensured that the fluid velocity matches that of the control particles without unnecessarily disturbing the natural small-scale fluid motion. To achieve this goal the overall fluid motion is separated from fine-scale detail using a low-pass filter.

ter on the current velocity field. Velocity forces are then computed with respect to the smoothed fluid velocities. The smooth velocity field  $\tilde{\mathbf{v}}_f$  is obtained using an approximation of discrete convolution with the kernel  $W$  of the control particles:

$$\tilde{\mathbf{v}}_f(e) = \frac{\sum_i \tilde{\mathbf{v}}_i W(d_{i,e}, h)}{\sum_i W(d_{i,e}, h)} \quad \text{with} \quad \tilde{\mathbf{v}}_i = \frac{\sum_e \mathbf{v}'_f(e) W(d_{i,e}, h)}{\sum_e W(d_{i,e}, h)},$$

where the filtered velocity for each control particle  $\tilde{\mathbf{v}}_i$  is computed with the current fluid velocities  $\mathbf{v}'_f(e)$  given by the simulation. The process of evaluating these two equation is illustrated in Figure 9.3. Note that  $\tilde{\mathbf{v}}_f$  is computed with velocities of the control particles, while  $\tilde{\mathbf{v}}_i$  is computed with a sum of fluid element velocities. The smoothed fluid velocity  $\tilde{\mathbf{v}}_f(e)$  then replaces  $\mathbf{v}(e)$  in Equation (9.5).

To show that this new control force only modifies the low-frequency part, while retaining the high-frequencies, the control force  $\mathbf{f}_c(e)$  is integrated separately, yielding  $\mathbf{v}'_c = k(\mathbf{v}_p - \tilde{\mathbf{v}}_f)$ . Here  $\mathbf{v}_p$  is the interpolated velocity of the control particles at a fluid element  $e$  and  $k$  is a constant depending on the user parameter  $w_v$  from Equation (9.5). By decomposing  $\mathbf{v}'_f$  into the low-pass filtered velocity  $\tilde{\mathbf{v}}_f$  and the high-frequency part  $\Delta\mathbf{v}_f$ , i.e.  $\mathbf{v}'_f = \tilde{\mathbf{v}}_f + \Delta\mathbf{v}_f$ , the new fluid velocity is given by

$$\begin{aligned} \mathbf{v}' &= \mathbf{v}'_f + \mathbf{v}'_c \\ &= \tilde{\mathbf{v}}_f + \Delta\mathbf{v}_f + k(\mathbf{v}_p - \tilde{\mathbf{v}}_f) \\ &= (1 - k)\tilde{\mathbf{v}}_f + k\mathbf{v}_p + \Delta\mathbf{v}_f. \end{aligned} \quad (9.7)$$

Hence, the low frequency part of the fluid velocity is blended with the velocity of the control particles, while the high-frequency part is retained, as sketched in Figure 9.4. The bottom row of Figure 9.1 shows the effect of the scale-separated force control that significantly better preserves fine-scale fluid motion.

To fine tune the effect of the velocity control, another parameter can be introduced to linearly blend between direct and detail-preserving control. Using a weight slightly larger than one, this method can even be used to artificially increase and reinforce the fluid details. This technique can be used similar to the vorticity confinement and vortex particle methods of [SU94, NFJ02] and [SRF05].

## 9.4 Results

For the implementation, the control particles are rasterized to the LB grid using early reject tests that prevent unnecessary evaluations of the influence forces. Due to the small changes of a single LB step it is sufficient to update the control force array in intervals. For the simulations presented here the forces are updated every 32 LB steps. To include the control forces into the LBM, the equilibrium DF is computed with the modified fluid velocity. Equation (3.4) for the collision is thus changed to

$$f_i(\mathbf{x}, t + \Delta t) = (1 - \omega)f_i^*(\mathbf{x}, t + \Delta t) + \omega f_i^{eq}(\mathbf{v} + \mathbf{f}(\mathbf{x}), \rho). \quad (9.8)$$

The effect of the detail-preserving control approach is shown in Figure 9.6. A column of fluid splashes against a wall at a T-junction. The behavior of the fluid without control can be seen in Figure 9.5, where the fluid flows symmetrically to both sides of

the junction. Figure 9.6 shows two simulations where a coarse set of 216 control particles forces the fluid to flow only towards the left. As can be seen in the upper two pictures, the direct velocity control introduces artificial viscosity, smoothing out the turbulence and vortices of the flow. The pictures in the lower row show the resulting fluid motion using the detail-preserving approach described in Section 9.3. The controlled flow with detail-preservation retains small-scale fluid features and therefore yields a more natural and interesting behavior. The simulation was performed with a  $240 \cdot 120 \cdot 120$  grid resolution, which took 38s per frame on average (without rendering) on a standard Pentium IV 3 GHz PC. The computation of the control forces took 2 – 4% of the total computation time.

In Figure 9.7 the fluid is forced to flow up several stairs and form a human figure. For the first part of the animation, 500 control particles are used. These are generated from a time-reversed coarse simulation of fluid flowing down the stairs. As the fluid reaches the upper platform, these control particles are blended with 5k control particles sampled from a 3D model of the human figure. The simulation was performed on a  $300^3$  grid resolution and took 142s per frame, including on average 4s for computing the control forces. An example of target shape matching, can be seen below in Section 12, Figure 11.5 and Figure 11.6.

These examples demonstrate the reduced artificial viscosity of the detail-preserving control. As the width of the influence radius of a control particle is coupled to the filtering of the velocity field, the scale of detail preservation is determined by the number of control particles. This allows large scale control with a low number of control particles, while additional sets of finer control particles can be used to modify smaller scales. In the future, the method could be extended to include anisotropic influence kernels, which could allow finer control with fewer control particles. The framework could also be used to control the deformation of elastic bodies, similar to [KK05]. Furthermore, it would be useful for practical applications to determine the influence parameters directly, e.g., from the motion of a target shape. This could help users to more easily achieve a desired fluid motion.

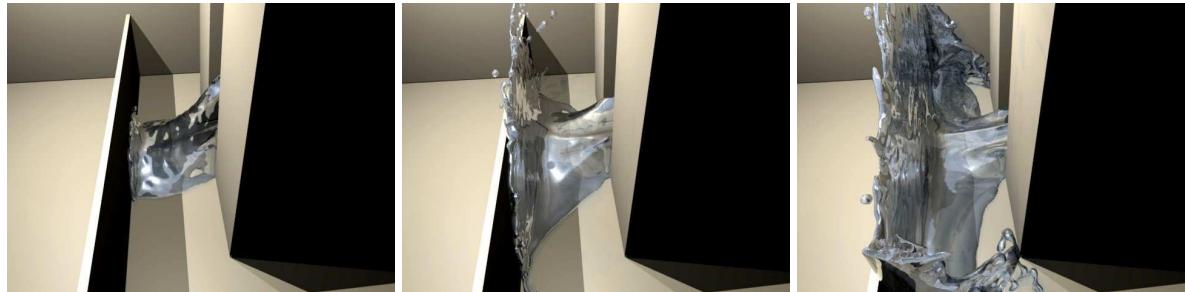


Figure 9.5: The uncontrolled breaking dam simulation. The water splashes to both sides at the T-junction.



Figure 9.6: Comparison of direct velocity control (left column) and detail-preserving control (right column) for the T-junction breaking dam.

---

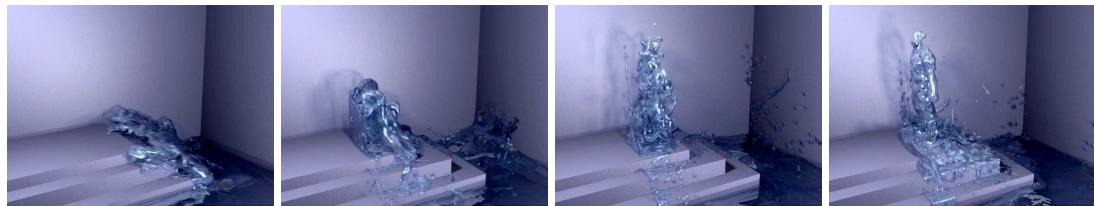


Figure 9.7: A fluid simulation is controlled to flow up the stairs and form a human figure.

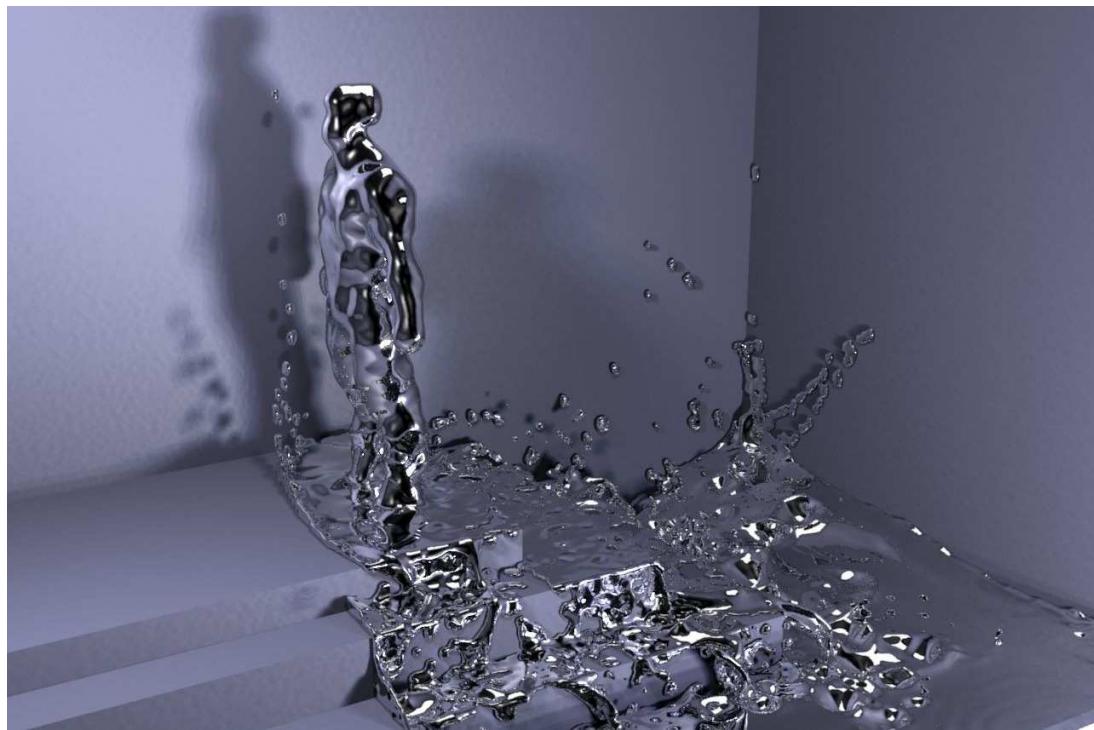


Figure 9.8: An image from the sequence of Figure 9.7 with a reflective surface visualization.

---



# Chapter 10

## Modelling Large Scale Fluids

The following section will focus on scenes that have a much larger physical scale than the scenes considered before, e.g. a ship traveling through the ocean. The challenge is to capture both the large scale movement of the water around the ship, as well as the splashing of the waves around it, including drops and spray. As all these phenomena contribute to the visual appearance, they have to be captured to achieve a realistic representation of such a scene in a computer generated animation.

For all classes of algorithms that are used to simulate free surface flows, the problem is that the amount of computational work and the required resources grow significantly when the resolution of the simulation is increased. The full simulation of the scales mentioned above with a VOF Navier-Stokes solver would hardly be possible even on large supercomputers. Adaptive techniques can be used to alleviate this problem to some extent, as shown in Section 7 or in [LGF04], but usually increase the complexity of a solver and have limits in their ability to speed up the computational time. In the following, a different approach will be presented that computes the full fluid flow only in a bounded region of interest, and uses a fast two-dimensional fluid simulation to compute the fluid surface around it. Only an upper layer of fluid is simulated, instead of the whole depth of the fluid from the free surface to the bottom (e.g. the ocean floor). The small scale details such as drops are simulated as particles with a simplified, yet physically based, algorithm.

In its different forms the *Navier-Stokes* (NS) equations have long been used for physically based animation. [KM90] were the first to use *shallow water simulations* in com-

puter graphics. As of today, this simplified model, which assumes depth averaged fluid properties, is still a research topic in computer graphics e.g. for computations performed on the GPU [HHL<sup>+</sup>05]. In other application fields, such as coastal engineering, it has become an important tool. For modelling large scale water surfaces deep-water wave models are often used to compute the water surface motion, e.g. in [Tes04] or [HNC02]. These, however, do not solve the NS equations directly, but model the wave propagation by different forms of trochoid function spectra. In [RNGF03], Rasmussen et al. combined a 2D flow field with periodic 3D simulations to create detailed smoke simulations. Large scale open water scenes with 2D and 3D simulations were also used in TV and feature film productions, however, without giving detailed information about the models used. Recently, an approach to optimize a fluid simulations with tall and thin cells was proposed by [IGLF06], thus also reducing the computational complexity for large fluid volumes. As the shallow water equations represent an advection-diffusion problem similar to the full NS equations, they can likewise be solved with the LBM. A derivation of the appropriate changes to the basic algorithm can be found in [Del01].

A different approach for detailed and accurate fluid simulation solvers that will be used in the following sections, can be found in the area of chemical engineering. For cases such as bubble column reactors, Eulerian-Lagrangian simulations of these dispersed multi-phase flows, e.g. large numbers of bubbles in a relatively coarse fluid flow simulation [DKvS99], are used to understand and optimize the physical processes [BGD05]. These methods simulate bubbles with a spherical shape, and model the forces caused by the turbulent fluid around them. Apart from level set methods, where particles are used to accurately track the free surface, [TFK<sup>+</sup>03a] also use particles to add small scale details. But in contrast to the approach presented in the following, they generate drops based of the surface curvature, and apply linear damping to model air resistance. More recently, Kim et. al presented techniques to model and render turbulent water with particle based level set solver [KCC<sup>+</sup>06]. This work, however, is also focused on scenes with a smaller scale.

## 10.1 Shallow Water Simulation

Within a certain region of interest, e.g. around a moving ship, a full three-dimensional simulation of the free surface fluid is performed. In this region the algorithm as described so far can be applied. The outer water surface is computed by solving the shallow water equations. Also known as St. Venant equations, they are usually used to simulate waves whose wavelength is similar to the overall water height. In this case the wave propagation speed is constant for all amplitudes. Deep water waves on the other hand are dispersive, which means that the wave propagation speed depends on their amplitude. By using a shallow water model the assumption is made that for the limited range of amplitudes generated by the three-dimensional simulation the wave propagation speed is the same. The advantage of a shallow water simulation is a full flow field for the water surface that can produce vortices or handle e.g. flowing rivers.

Shallow water simulations (SWS) can likewise be performed using the LBM. In this case, instead of considering the fluid pressure, a height value is computed for each cell.

---

Overall, the algorithm is very similar to the basic algorithm described above – both the streaming step and Equation (3.4) for relaxation towards the equilibrium are still valid. The equilibrium DFs to be used with Equation (3.4), however, are calculated differently. Furthermore, as the fluid surface is only two-dimensional, the *D2Q9* LB model with nine velocities is used. To distinguish the DFs of the shallow water simulation from those of the three-dimensional free surface simulation, they are denoted as  $g_l$  in the following. The fluid height  $h$  and the fluid velocity for the shallow water simulation are calculated as

$$h = \sum_{l=1}^9 g_l \quad \mathbf{v} = \frac{1}{h} \sum_{l=1}^9 \mathbf{e}_l g_l . \quad (10.1)$$

In contrast to the 3D LB model, the velocity computation of the SWS requires a division by the height, as shown in Equation (10.1). With height and velocity, the equilibrium DFs are computed as

$$g_0^{eq}(h, \mathbf{v}) = h \left[ 1 - \frac{5}{6} Gh - \frac{2}{3} \mathbf{v}^2 \right] , \quad (10.2)$$

and

$$g_l^{eq}(h, \mathbf{v}) = w_l h \left[ \frac{1}{6} gh + \frac{1}{3} \mathbf{e}_l \cdot \mathbf{v} + \frac{1}{2} (\mathbf{e}_l \cdot \mathbf{v})^2 - \frac{1}{6} \mathbf{v}^2 \right] , \quad (10.3)$$

for  $l = 1..9$ . Here  $G$  is the gravity force, normal to the two-dimensional plane of the SWS, and the weights  $w_l$  have the values  $w_l = 1/18$  for  $l = 2, \dots, 5$ , and  $w_l = 1/36$  for  $l = 6, \dots, 9$ . An in depth description of the shallow water LBM can be found in e.g.

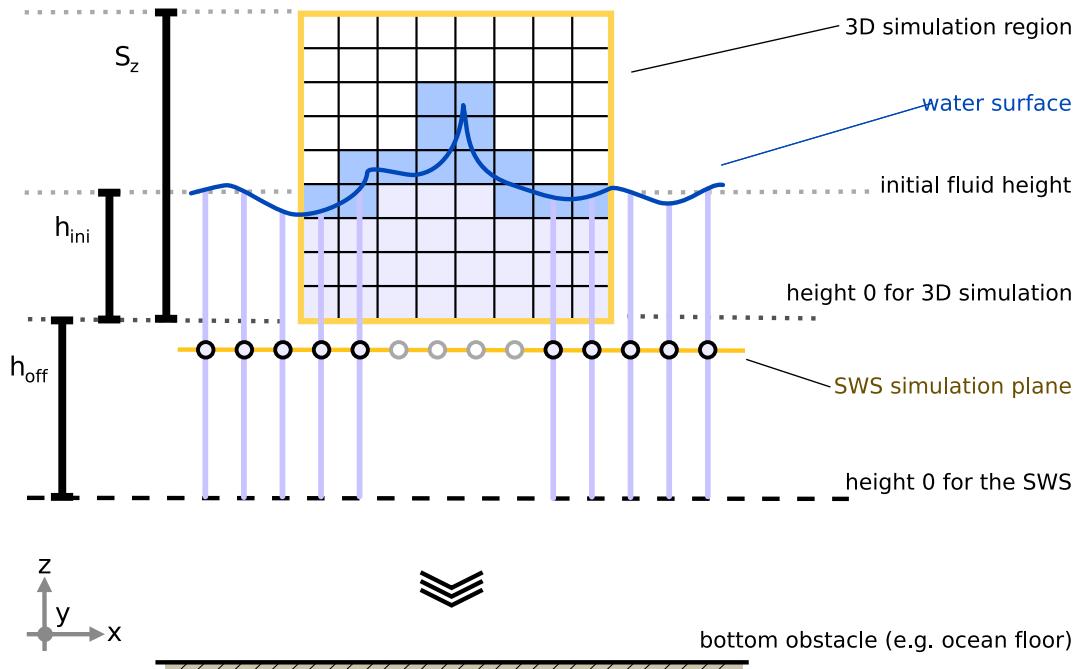


Figure 10.1: This picture gives an overview of the hybrid simulation method. The full three-dimensional fluid flow is solved in a given region of interest (illustrated by a 2D rectangle), and coupled to a two-dimensional shallow water simulation (shown as a 1D line in the picture).

[Zho04]. To establish a fixed height of the SWS boundary, the cells there are set to have the equilibrium DFs for the initial height and a zero velocity.

As this shallow water solver is similar to a basic LB solver, the Smagorinsky turbulence model from Section 3 can likewise be used to increase stability. The only difference is that  $\Pi_{\alpha,\beta}$  is now computed as a sum over the nine DFs of a SWS cell. To ensure stability for varying velocities, the adaptive time stepping as described in Section 6 is applied to the SWS simulation as well.

## 10.2 Hybrid 2D/3D Simulation

An overview of the hybrid simulation approach is given in Figure 10.1. Both algorithms have been parametrized to solve the same fluid simulation problem, and are then coupled at an interface region. In the following it will be assumed that the SWS is performed in the  $xy$  plane, and the gravitational force acts in the direction of the negative  $z$  axis. There is an inherent difference between the two simulation approaches that has to be overcome: the derivation of the SWS assumes a depth averaged velocity and has a coupling between fluid height and velocity. The 3D simulation, on the other hand, can have a velocity varying along the  $z$  axis, and has boundary conditions (see below) that makes it independent of the initial height of the fluid surface  $h_{\text{ini}}$ . In order to be able to couple both simulations, the parametrization procedure explained in the following for the SWS was developed. It ensures that the SWS has the same wave propagation speed as the average waves generated in the 3D simulation.

The SWS is offset by a constant height  $h_{\text{off}}$ , as shown in Figure 10.1. Here  $S_z$  is the height of the 3D domain in cells. In combination with the gravity the height offset  $h_{\text{off}}$  determines the wave propagation speed, and is set according to the average wave height generated by the 3D simulation  $h_{\text{avg}}$ . Assuming a common trochoid wave shape,  $h_{\text{off}}$  is set to be half of the expected wave length, thus  $h_{\text{off}} = \pi h_{\text{avg}}$ . For the examples shown in the following a value of  $h_{\text{avg}} = 1/2h_{\text{ini}}$  was used. Now the gravity force of the SWS has to be scaled according to the height offset. This is done by examining the behavior of the SWS properties. Given an arbitrary simulation setup, the properties of the fluid change by a factor given in Table 10.1, when the value of the parameter in the first column is multiplied by 2. Thus, given the initial SWS fluid height and

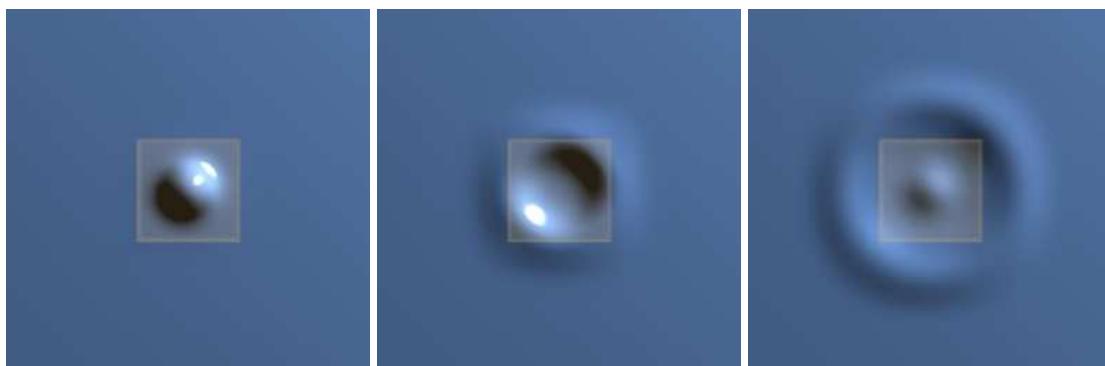


Figure 10.2: Wave propagation of a hemispherical drop on a flat surface using the SWS coupling algorithm. The 3D simulation region is highlighted in the middle.

$n = \log_2(h_{\text{off}} + h_{\text{ini}})$  the SWS gravity  $G$  is set to match the given offset for the simulation resolution. It is computed from the z component of the 3D gravity  $\mathbf{g}_z$  as

$$G = \mathbf{g}_z \cdot \left(\frac{e}{2}\right)^{-n}. \quad (10.4)$$

The 3D gravity is given by the physical value, usually  $G' = 9.81 \text{ [m/s}^2]$ , as  $\mathbf{g} = (0, 0, G' \cdot \Delta t^2 / \Delta x)$ . The initial time step size  $\Delta t$  is set according to the maximum fluid or moving obstacle speed in the simulation setup. Now, when transferring velocities in the xy plane between the simulations, the influence of the offset and gravity scale have to be removed. According to Table 10.1 this is accomplished by

$$\mathbf{v}_{x,y} = s_u \mathbf{u}_{x,y}, \text{ with } s_u = \frac{\sqrt{2}^n}{\sqrt{s_g}^{1/n}}. \quad (10.5)$$

### 3D to 2D Coupling

Here, the height of the fluid at a position within the 3D simulation region is determined by searching for the first interface cell. This search is started at the cell with grid position  $(i, j, 0)$ , and assumes a planar fluid surface. Hence, the fluid height is computed for the first interface cell at  $(i, j, k_H)$  with

$$H(i, j) = k_H + \epsilon(i, j, k_H) + h_{\text{off}}. \quad (10.6)$$

The velocity at the water surface  $\mathbf{u}_{\text{surf}}$  is given by the interface cell of the 3D simulation. To transfer the information from the 3D simulation to the SWS a cell at  $(i, j)$ , shown as a circle marked with X in Figure 10.4, is initialized with the equilibrium DFs

$$g_l = g_l^{eq} \left( H(i, j), s_u \mathbf{u}_{\text{surf}} \right). \quad (10.7)$$

The cells where height and velocity are set with Equation (10.7) represent the inner boundary for the SWS. Further inwards the full 3D simulation is performed, thus the cells of the SWS do not have to be updated in this region, as their values are never used. To ensure a transfer with as few disturbances as possible, a double layered transfer is used. Thus, a second type of boundary condition for the region of SWS cells

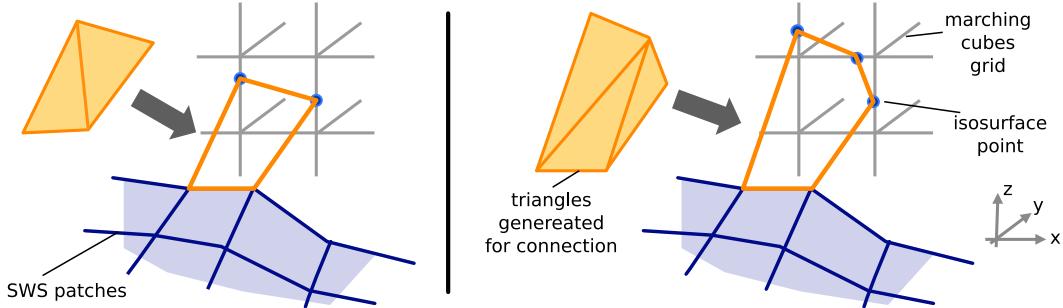


Figure 10.3: The two cases that need to be distinguished to generate a closed surface mesh for the 3D and shallow water simulations.

Table 10.1: Behavior of the SWS on parameter change. Here  $N$  is the number of SWS cells along the x or y axis.

	$G$	$N$	$\Delta t$	$v$
$G$	2	1	$\sqrt{2}$	$\sqrt{2}$
$N$	1	2	$e$	$\sqrt{2}$

directly outwards of the boundary cells described above (the circle marked with O in Figure 10.4) is applied. For the cells that are updated according to Equation (10.7) all DFs are reset each time step, while for the second boundary layer the existing DFs are only rescaled to match the required fluid height

$$g_l^* = g_l \cdot \frac{H(i, j)}{h(i, j)}. \quad (10.8)$$

The combination of these two boundary conditions ensures a correct transfer of both fluid surface height and velocity.

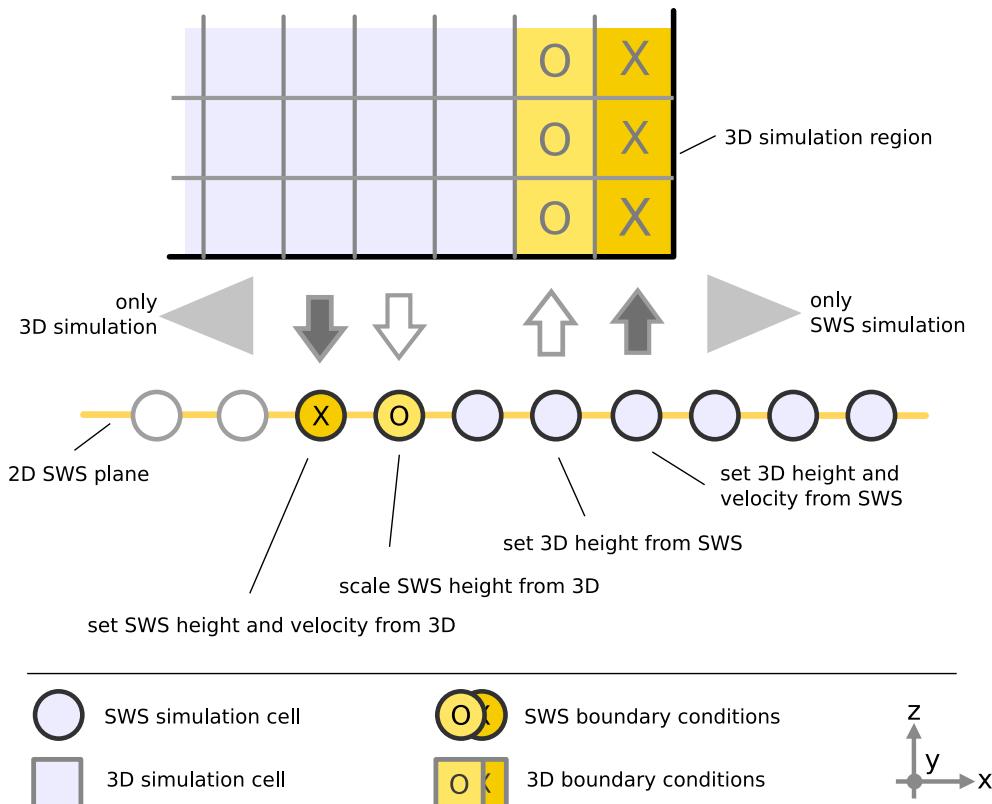


Figure 10.4: Detail of the double layer boundary conditions in the overlapping interface region.

## 2D to 3D Coupling

The transfer from the SWS to the 3D simulation, is done by initializing the 3D cells to represent the SWS height and velocity. For a 3D cell at  $(i, j)$  marked with O in the upper part of Figure 10.4 cells are thus removed if  $H(i, j) > h(i, j)$ , otherwise new ones are added. To correctly initialize the new cells the velocity of the SWS is directly used. The z coordinate of the velocity is calculated from the SWS fluid heights of the current and previous timestep.

For the outer layer of the 3D simulation (square cells marked with X in Figure 10.4) velocity boundary conditions with a fixed pressure are used. For the LBM the pressure of a cell with height  $k$  in the domain is given by

$$\rho_k = 1.0 + (h_{\text{ini}} - k) \cdot g_z \cdot -3\omega . \quad (10.9)$$

The pressure thus increases further down in the grid, with a gradient that depends on the relaxation time  $\omega$ . The velocity can again be taken directly from the SWS, as described above. Note that it is in this case not necessary to scale the SWS velocities, as the whole height of the 3D simulation is set. These boundary conditions, however, do not ensure the full propagation of arbitrary waves generated in the SWS region, as this would require an additional wave profile initialization. Although these boundary conditions do not enforce mass conservation for the transfer, this is not problematic as the overall height of the fluid is kept at the initial value by the SWS and the pressure initialization of the 3D simulation.

The depth of the overlapping region for the two simulations is variable, but a distance of one eighth of the 3D domain size was found to yield good results. A validation run is shown in Figure 10.2. The circular wave retains its shape while it is transferred from the highlighted 3D region to the SWS region. The in- and outflow at the 3D domain boundary furthermore causes no disturbances of the flow field. A pure SWS simulation would not have been able to resolve the drop forming in the middle of the 3D region, visible in the right picture of Figure 10.2.

## Surface Generation

As described before, the triangulation of the 3D simulation surface is performed with the marching cubes algorithm [LC87]. A triangulation of the SWS surface is easily



Figure 10.5: Effect of the drag model for different size scales. The simulations were parametrized to represent scales of 10cm, 1m and 10m, from left to right. The smaller particles, are slowed down, and cause mist below the outflow.



Figure 10.6: A stream of water hits a rock and a water surface In the rightmost picture the interface between 2D and 3D region is highlighted.

computed by constructing patches between four adjacent SWS cells that have  $x$  and  $y$  coordinates according to their grid position, and a  $z$  coordinate given by  $h(x, y)$ . At the 3D domain boundary the first row of SWS cells is left out, and new triangles are constructed to connect the 3D mesh to the SWS patches. If both points of a marching cubes cell lie on its  $z$  edges, this is sufficient to ensure a closed mesh. For all other cases, triangles also have to be connected to the points above or below the cell at the surface, as shown in Figure 10.3. In rare cases, e.g. when a drop directly hits the connection line, this technique will not result in a closed mesh. In the interface region, where full information is available from both simulations, the fluid surface heights are linearly blended, to achieve a smooth transition from one type of simulation to the other. As the mesh generated from the fluid fractions already requires smoothing, a smoothing of the interface region is also performed to prevent any artifacts from misaligned normals.

### 10.3 Lagrangian Drop Model

For the animation of drops methods developed for dispersed gas-liquid flows are applied. Each drop is described by its position  $\mathbf{x}$ , velocity  $\mathbf{w}$  and radius  $r$ . It is assumed that the drops are small enough to remain spherical due to surface tension. Thus using the density of water  $\rho_W$  the mass of a drop is given by its volume

$$m_P = \rho_W \frac{4}{3} \pi r^3. \quad (10.10)$$

## Generation

To generate particles in the simulation the turbulence model explained in Section 10.1 is used. As it already determines the amount of unresolved flow features in a given cell, it is also used to compute a particle generation probability for cells at the fluid interface. This probability is calculated with the absolute value of the Reynolds stress tensor given by Equation (3.6) and the physical speed  $\mathbf{u}' = \mathbf{u} \Delta x / \Delta t$ . The magnitude of the stress tensor usually takes values of ca.  $P_m = 10^{-2}$  for regions with significant unresolved detail independent of the actual grid resolution. It is assumed that the range of velocities, where the pressure of the surrounding air causes instabilities that lead to drops at the surface, is similar to that of the drop terminal velocities, which motivates the following probability function

$$p_D = P_{ab} * (\mathbf{u}')^2 \quad \text{with} \quad P_{ab} = |\Pi_{\alpha,\beta}| . \quad (10.11)$$

Thus for a high physical speed of  $10[m/s]$  and significant unresolved flow details this function will result in a drop generation probability close to one. Note that the calculation of the Reynolds stress tensor is especially easy for the LBM, as it is computed locally from the derivative information contained in the non-equilibrium parts of the DFs (see Equation (3.6)). For other types of solvers, this computation will require access to neighboring grid cells to compute the derivatives. On creation the drop velocity is initialized with the fluid velocity and a randomized normal offset to avoid immediate collision with the fluid surface.

## Animation

For each LB step, the particle positions are updated using their velocities and the LB time step length

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \mathbf{w} . \quad (10.12)$$

To update the particle velocities, the balance of the forces acting on it is computed

$$m_P \frac{d\mathbf{w}}{dt} = F_G + F_D . \quad (10.13)$$

where  $F_G$  is the force due to gravity and  $F_D$  is the drag force caused by the drop of water moving through the air. In contrast to the dispersed flow simulations mentioned above, any lift the drops might experience is thus ignored, as well as other forces that would e.g. be caused by the density gradient in the air. The lift is proportional to the ratio between the involved fluids, which is close to zero for air and water. Likewise, a density gradient very close to zero is assumed for the air phase.  $F_G$  is directly computed from gravity and particle mass as  $F_G = m_P g$ , while the computation of the drag force requires more effort. The movement of water drops through the air has been studied in depth for meteorological purposes, see e.g. [PK97]. From these studies it is known that rain drops usually have a size less than  $4.5mm$ . Above this size they will start to deform during their movement and eventually break apart due to the high forces from the air in comparison to the surface tension. It was furthermore measured that these large drops have a terminal velocity of up to  $w_{t1} = 9m/s$ , while smaller drops of with e.g.  $r = 0.5mm$  only accelerate to ca.  $w_{t2} = 2m/s$ . Given a coefficient of

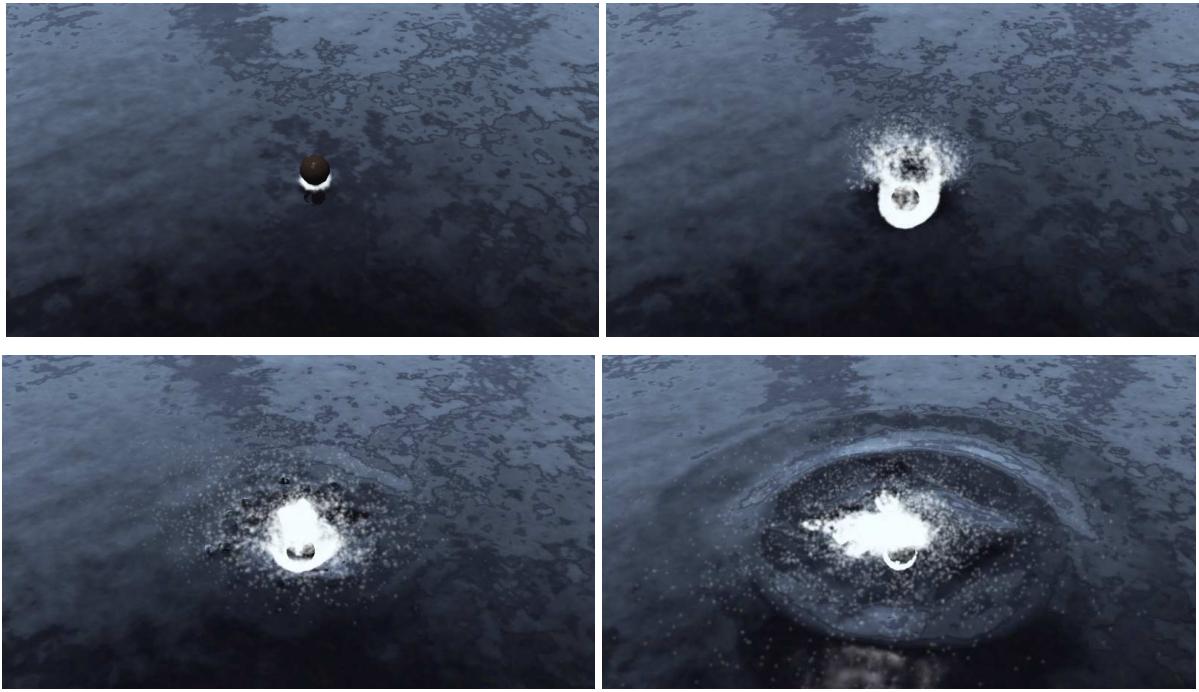


Figure 10.7: Animation of a spherical object hitting the water surface.

drag  $C_D$ , the drag force acting on a particle is calculated as

$$F_D = \frac{C_D}{2} \rho_L \pi r^2 \mathbf{w}_{\text{rel}} |\mathbf{w}_{\text{rel}}| , \quad (10.14)$$

where  $\mathbf{w}_{\text{rel}}$  is the relative velocity of the particle. It is computed from the velocity of the air  $\mathbf{w}_A$  by  $\mathbf{w}_{\text{rel}} = \mathbf{w}_A - \mathbf{w}$ . As the gas phase is not explicitly simulated,  $\mathbf{w}_A$  is usually set to zero. Other values could be used to e.g. simulate the effect of wind. For the drag coefficient there are various approximations for different regimes of turbulence. As the case of a larger spherical rain drop at terminal velocity is already turbulent ( $Re > 1000$ ), and the approximations are computationally very expensive, the computation of the drag coefficient is constructed to yield values in the required range. It is required that the drag force and gravity acceleration balance for the drops at their respective terminal velocities. Assuming a linear change for both parameters, the drag force is computed with a bilinear interpolation

$$C_D = \frac{|\mathbf{w}_{\text{rel}}|}{w_{t2} + (w_{t1} - w_{t2})w_r} \left( \frac{1}{2} + \frac{1}{2}w_r \right) . \quad (10.15)$$

with  $w_r = (r_1 - r)/(r_1 - r_2)$ . For the simulations the size of the drops was limited to the range of  $r_1 = 0.005m$  to  $r_2 = 0.0005m$ . After the computation of  $F_G$  and  $F_D$  the velocity is updated according to Equation (10.13) with an Euler-step

$$\mathbf{w}(t + \Delta t) = \mathbf{w}(t) + (F_G + F_D)m_P \Delta t . \quad (10.16)$$

## Additional effects

To actually cause a disintegration of a thin fluid sheet into drops, a size  $r_D$  in the given range is randomly chosen, and the mass of the drop subtracted from the interface cell

where it was generated. For simulations representing a large scale, this could result in huge numbers of particles – for these cases a multiple of the drop’s mass is subtracted from the cell. It is then displayed as a correspondingly larger transparent particle, thus representing multiple drops of similar size. Once the drop hits a fluid surface, the mass that was subtracted before is added again. Here, similar to [TFK<sup>+</sup>03a], the particles are traced on the fluid surface to give the impression of foam. Figure 10.5 shows examples of the drag force influence for different scales. For the larger test cases the higher velocities result in higher drag forces, resulting in a noticeable slowdown of the smaller drops.

Another effect that cannot be directly simulated with the algorithm explained in Section 10.1, is that of instabilities caused purely by the relative physical velocity of the fluid  $\mathbf{u}'_{\text{rel}}$ , as the air is not simulated as a fluid itself. Thus, in order to cause these instabilities, a simple approximation is used to manually add the following term

$$f_i = f_i + (P_m - P_{ab}) \cdot w_i \frac{\mathbf{u}'_{\text{rel}}}{50} \cdot \mathbf{e}_i , \quad (10.17)$$

for cells, with  $P_{ab} < P_m$  that do not generate particles.

## 10.4 Results and Discussion

All results shown in the following were created using a physical viscosity of water  $\nu_W = 1 \cdot 10^{-6}$ . To enhance the realism, a texture is added to the water surface, giving the impression of smaller chaotic waves. A test case of the hybrid simulation method is shown in Figure 10.6, where a stream of water hits a rock and a water surface. The waves that are generated spread outwards without a visible border between the SWS and the 3D simulation. Simulation resolutions and times can be found in Table 10.2. A test case that demonstrates the capabilities of the drop model is shown in Figure 10.7. A spherical object is dropped into a fluid surface. The drop model, with up to 50000 drops at a single time step, enhances the impression of a large simulation scale.

Given a working hybrid simulator, only small changes are necessary to achieve animations such as shown in Figure 10.8. Here the 3D domain is moved according to the position of an object in the xy plane. For each movement of the domain by  $\Delta x$  the values stored in the grid are copied by one in the desired direction. During the next step, the boundary regions of both simulations will again be correctly initialized for the boundary conditions. In this example the foam particles on the water surface clearly visualize the flow field in the shallow water region.

For a simulation run with a relatively large SWS domain, such as shown in Figure 10.8, the workload distribution between the different parts of the algorithm was measured. In this case, ca. 68.8% are spent on the simulation of the 3D region. The 2D region, covering a 35 times larger area, requires 24.9% of the time, while ca. 2.6% are spent on the coupling of both simulations. The remaining 3.7% were spent on drop calculation, surface mesh generation and initialization. The update of a single SWS cell is on average three times faster than the update of a single 3D cell.

In the future, it might be useful to apply the methods from SPH to the generated drop particles for an intermediate scale. This could be used to e.g. accurately capture

Table 10.2: This table shows grid resolutions together with average simulation times per frame (measured with an 2.2 GHz Opteron CPU). The size value is the physical length of a side of the 3D domain used for parametrization of the simulations.

	3D	SWS	Simulation	Size
Figure 10.6	$120^3$	$480^2$	14.6 s	0.2 m
Figure 10.7	$120 \times 120 \times 200$	$480^2$	34 s	10 m
Figure 10.8	$120^3$	$960^2$	79.6 s	2 m

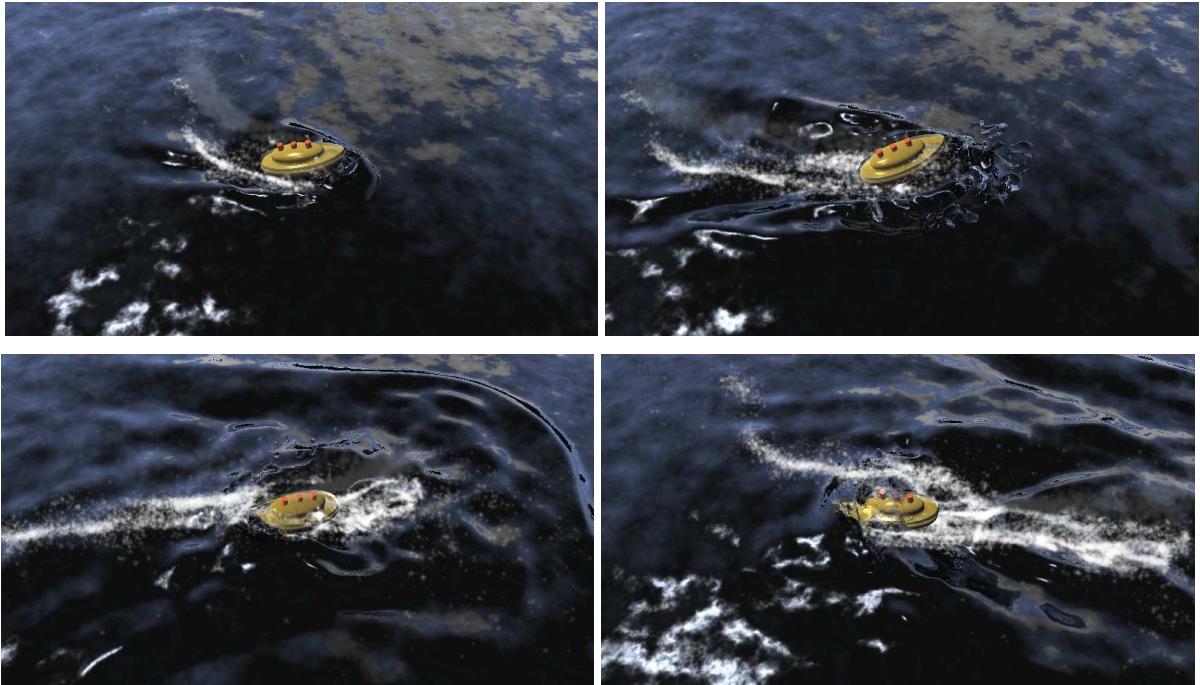


Figure 10.8: Pictures from an animation with a moving 3D domain.

effects such as coalescence. It would furthermore be interesting to add a model for the generation of drops in the SWS region as well, or couple it with an FFT solver for ocean waves [Lov03, Tes04]. An easy way to further speed up the computations would be to reduce the SWS resolution by an integer factor, and interpolate the values at its boundary. The problem of a fixed wave propagation speed, on the other hand, could be alleviated by overlaying multiple shallow water simulations with different parametrizations. Finally, the method could be used to couple multiple regions of three-dimensional computation in one large water surface simulation. Given enough computational resources in combination with low grid resolutions, this could be used to simulate interactive environments with large water surfaces e.g. for virtual reality applications.



# Chapter 11

## A Programming Interface for Fluid Solvers

### 11.1 Blender Integration

This chapter will explain how to construct an application programming interface (API) between 3D applications, such as CAD or animation programs, and fluid simulators, such as the one that was implemented for this thesis. An exemplary integration of the fluid simulator with its API is available as part of the open source 3D application *Blender*, an overview can be found in [Thu06]. It was published under an open source

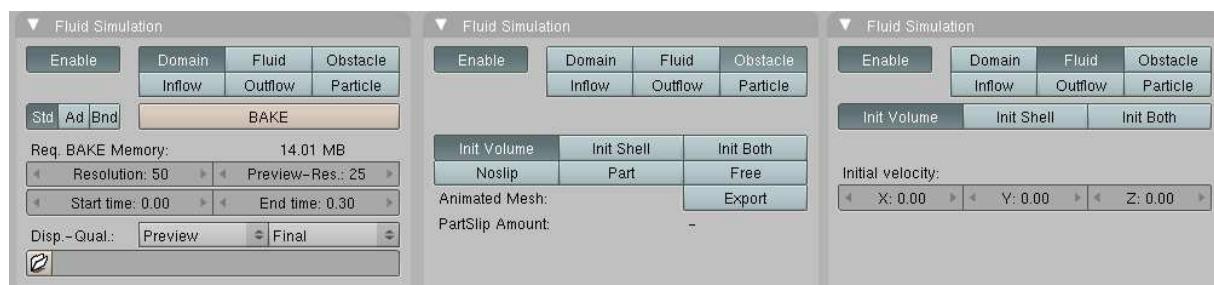


Figure 11.1: An example of three different setting panels from the Blender GUI.

license, and is available from [www.blender.org](http://www.blender.org) together with a manual and several tutorials.

Overall, the programming interface can be held very small – the application needs to pass parameters and geometry data to the solver, which generates a sequence of triangulated fluid surfaces together with additional information from the simulation. Several screenshots from the Blender GUI with options for different types of objects participating in a fluid simulation can be seen in Figure 11.1. A specific object type is designated to contain global information, among others, fluid viscosity and grid resolution, and set the boundaries of the fluid simulation domain in the coordinate system of the 3D application. Another object type defines obstacles in the simulation, which have various settings to control the obstacle surface. The volume of the mesh or its shell will then be initialized as obstacle cells for the simulation. Similarly, object volumes can be defined to contain fluid at the beginning of a simulation. For these, parameters such as the initial velocity are available. An example of a typical setup can be seen in Figure 11.4. As the GUI is usually specific to the 3D application, the main task here is to present all relevant options of the fluid solver in a clear and intuitive way. For example, the most commonly used options, such as grid resolution should be available without too many actions from the user.

Upon starting the simulation, the necessary data is transferred to the fluid solver. While this is straightforward for constant integer or floating point parameters, the geometry and time-varying parameters have to be converted into the format required by the solver API. Triangle meshes, for example, are specified by their vertices, and a set of triangle indices. Other information that is usually available in 3D applications, such as surface normals and properties can be ignored for the solver export.

As the simulation run can take large amounts of time, it is important to keep the user interface responsive. Hence, the simulation is started in a separate thread, while the GUI can still process user input. For the integration into Blender, the GUI is used to check for a signal from the user to stop the simulation, and display the current status of the simulation run. The simulation itself creates two compressed output mesh files for each animation frame – one in full resolution, and a smaller one with reduced resolution. The latter one can then be previewed quickly in the GUI, while the full resolution mesh may take several seconds to load from hard disc. A side by side comparison of the preview mesh, the full resolution mesh and a raytraced image of the full resolution

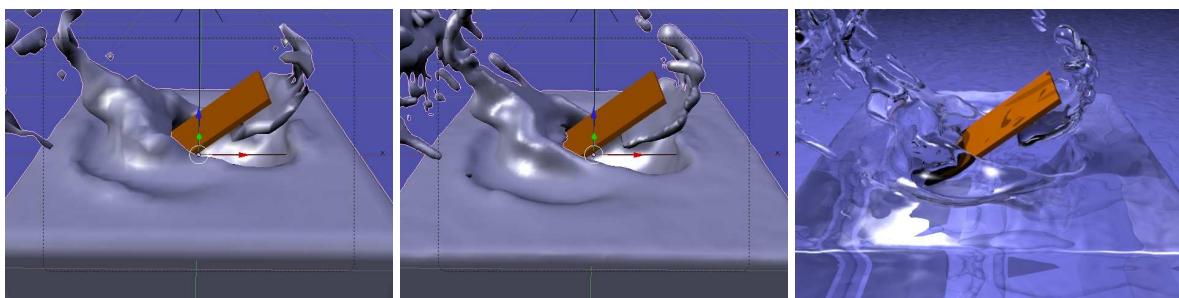


Figure 11.2: The image to the left shows a typical preview mesh in Blender. In the center the final mesh in the GUI can be seen, while in the right picture the final raytraced image is shown.

mesh is shown in Figure 11.2.

## 11.2 Integration Extensions

A smooth appearance of the fluid surface meshes is important for a plausible appearance. As subdivision of the triangulated mesh by itself does not give significantly better results, a simpler approach is to use smoothed vertex normals. These can be generated from the fill fraction field of the simulation. The difficulty of passing these normals to the raytracer strongly depends on its architecture. Blender usually uses normals that are generated from the actual triangle data. It is thus important to ensure that the smoothed normals from the simulation are not overwritten, unless other functions in Blender are applied to change the fluid surface geometry.

As motion blur is an important perceptual clue that is very hard to compute accurately and efficiently within a raytracer, many 3D applications apply image based motion blur techniques. These compute the velocities of each pixel similar to a usual raytracing pass, which computes the pixel color. The pixel velocities are given by the motion of the triangle that is hit by the corresponding ray. Each vertex of the triangle has a velocity that is interpolated for the intersection point on its surface. As the fluid surface can completely change from one frame to the next, the motions of the triangles and vertices cannot be calculated from the triangle meshes. However, the velocity information is available during the simulation. A per vertex velocity is thus exported together with each triangle mesh. During the motion blur pass, these are used and interpolated for each triangle of the fluid surface mesh. A comparison of an image from a fluid simulation with and without motion blur can be seen in Figure 11.3. While for the image on the left the fluid motion is not clearly visible from the still image, it is apparent from the right image of Figure 11.3. The latter one, however, has less visual detail in the blurred regions – the motion blur is thus especially suitable for animations.

Most images of simulations in previous chapters were set up using Blender – another example of a complex animated character interacting with controlled fluids can be seen in Figure 11.5 and Figure 11.6.

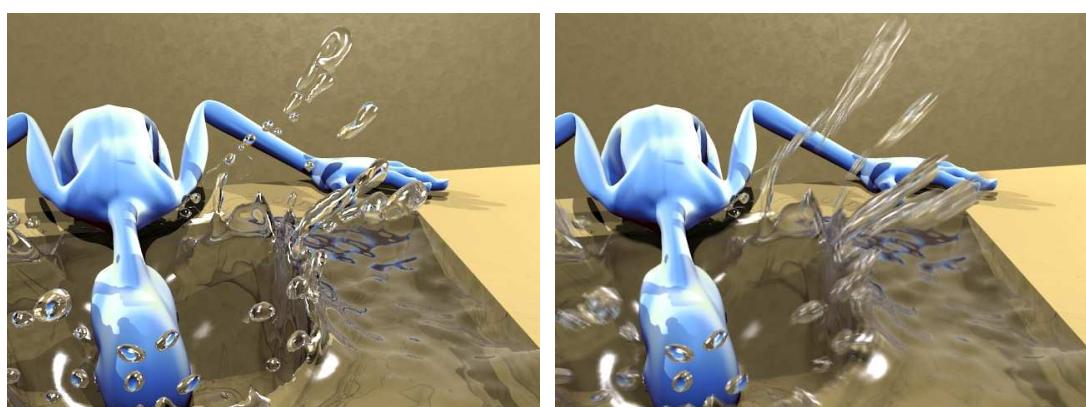


Figure 11.3: The left image shows a fluid simulation scene without motion blur, while the image to the right has an image based motion blur with per vertex velocities applied.



Figure 11.4: Several frames of animation from a simulation setup in Blender – a glass shaped obstacle is filled with fluid.

---

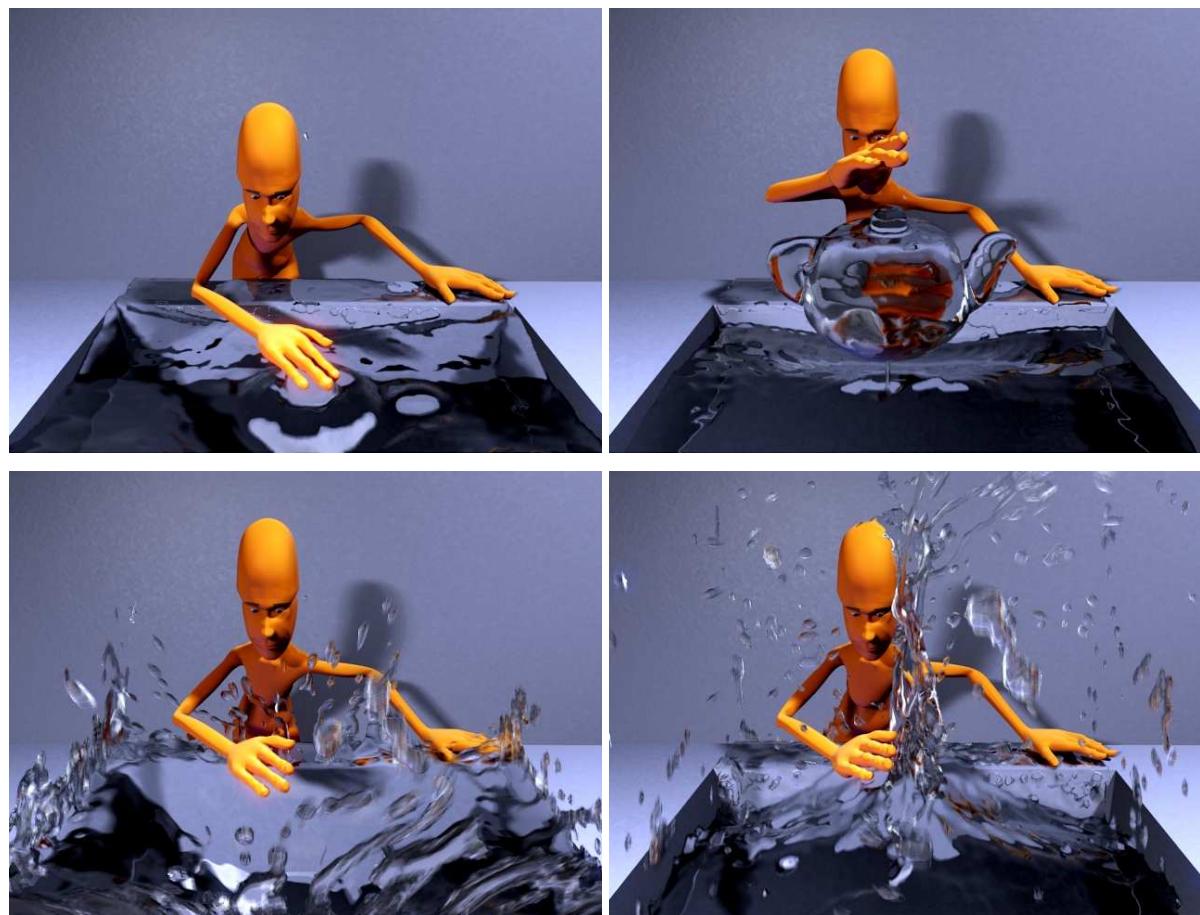


Figure 11.5: An example of an animated character in combination with a controlled fluid. Both were set up using Blender.

---

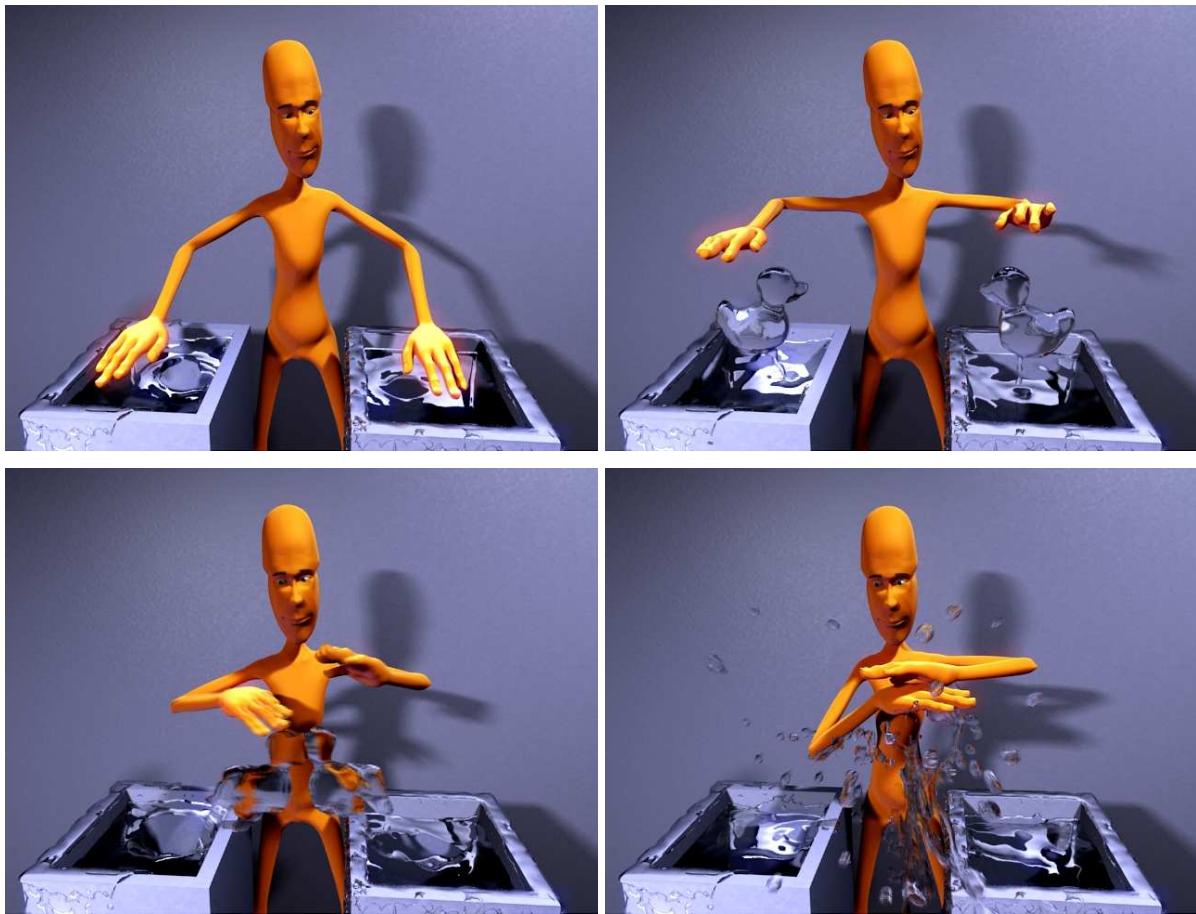


Figure 11.6: Another character and fluid control example animated and rendered with Blender.

---



# Chapter 12

## Conclusions

The goal of this thesis was to develop a free surface simulator with high efficiency, stability and flexibility. The following section will summarize the contributions of this thesis that were presented in order to achieve this goal. Afterwards, drawbacks and future extensions of the algorithm will be discussed.

### 12.1 Summary

The efficiency of the algorithm, as presented in this thesis, is given due to the use of the LB solver for the NS equations. The basic algorithm performs very well on modern computer architectures, and is suitable for shared- and distributed-memory machines. The VOF free surface model can be directly integrated into the LBM, and likewise yields a high efficiency. It preserves mass, and allows a local treatment of the free surface boundary conditions. Additionally, the efficient treatment of arbitrary moving and deforming triangle meshes as obstacle objects is important for the practical use of the algorithm.

The stability is mainly guaranteed by the adaptive time stepping method, and the turbulence model. The time step size is automatically chosen according to the largest velocities occurring in the simulation. It thus guarantees valid velocities even for highly dynamic scenes. The combination of the adaptive time steps with the Smagorinsky LES model allows the simulation to be performed with very small time step sizes

that ensure a stable simulation even for flows with very high velocities. Furthermore, the LES model guarantees the stability of the solver for low viscosities.

The central part of this thesis, the adaptive grid coarsening algorithm, contributes to both efficiency and flexibility of the simulator. It speeds up the computation times for large grid resolutions, and enables the practical use of high resolutions for animations. It is based on an established grid refinement scheme for the LBM, and extends this scheme with a set of cell based rules to ensure an adaptive coarsening of larger fluid volumes. At the same time valid boundary layers are guaranteed. The overhead caused by these cell checks causes no noticeable slowdown for smaller simulation resolutions, while allowing speed up factors of more than three for larger resolutions.

Moreover, the flexibility of the LB solver is increased by the detail-preserving control framework, the Lagrangian drop model, and the shallow water coupling. The control framework allows the creation of animations according to the wishes of the user – this is crucial for almost all types of animation production. The framework preserves the natural fluid motion, which is of particular importance for an interesting and realistic appearance. The coupling of a 3D free surface simulation with a shallow water simulation allows the animation of large open water surfaces that could not be simulated even with the adaptive coarsening technique due to memory and time requirements. In addition, the Lagrangian drop model can be used to enhance the realistic appearance of a scene by giving the impression of small scale drops and foam. Finally, the integration of the solver into a 3D application made the setup and visualization of complex fluid scenes possible.

## 12.2 Discussion and Future Work

One of the main strength of the algorithm developed in this thesis is the computational efficiency of its components. The computations usually involve only the local neighborhood of a cell. Therefore, global properties – such as the mass conservation or incompressibility – are guaranteed by the local treatment of the algorithm. Several aspects of the simulation can, however, still be improved and extended. The following sections will discuss areas of future work for animation applications, as well as two engineering applications: the production of metal foams and particle technology applications.

### Animation of Free Surface Fluids

For the animations that where simulated for this thesis the memory requirements where often a limiting factor, even though the *grid compression* technique of [Wil03, Igl05] was applied. The algorithm in its current implementation requires 22 floating point values to be stored per cell, which means that a full allocation of a  $256^3$  grid already requires almost 1.5GB of memory. This could be alleviated by dynamically allocating memory only for the regions of the grid that are filled with fluid. In order to retain a high performance, this could be implemented by handling grid patches of a fixed size, organized, e.g., in a space tree structure.

---

Another problem is caused by the underlying VOF free surface tracking, as its limited range of fill fraction values (zero to one near the interface) does not allow an accurate computation of the interface curvature. For an approximation of the curvature, this may be sufficient, but for simulations with strong surface tension, an inaccurate curvature calculation can lead to visible artifacts. The accuracy can be increased by allowing larger computational stencils, but it would be useful to develop methods that are able to efficiently and locally compute the curvature, e.g., along the approach proposed in [PP04].

Several aspects of the algorithm are interesting topics of future research. One possible extension is the close coupling between a LB, and a SPH solver, extending the Lagrangian drop model of Section 10. This could enable large scale fluid simulations that correctly handle all necessary scales – from large fluid volumes, to interacting splashes, to foam and spray. Due to the VOF method, which computes actual mass values for each cell, the mass conservation can easily be guaranteed for such a hybrid simulation.

For the coupling with shallow water simulations, dispersive wave models, such as the Green-Naghdi equations [DS05] could further enhance the results. As these equations are non-linear, they are significantly harder to solve, but could be used to handle different wave propagation speeds in the shallow water region. Another approach would be to use the convolution based algorithms, as proposed in, e.g., [Lov03], and by J. Tessendorf in [Kir04]. However, these would require more changes to the boundary conditions of the coupled simulations, as the methods mentioned previously do not yield velocity values for the fluid surface.

Moreover, it should be possible to couple multiple LB simulations, as demonstrated in, e.g., [MSRG05, LIG06, LSSF06] for SPH and level set based solvers. This would yield interesting visual effects, and increase the suitability of the solver for practical fluid animations. Smoke and fire simulations themselves are possible with the LBM, as demonstrated, e.g., by [WLMK04]. An additional density value is advected in the fluid velocity field given by the simulation of the gas phase. This gas phase could then be coupled to the free surface simulation, using the free surface as a moving boundary condition, possibly within the same LB grid as the free surface simulation.

Another area with room for improvements is the overall usability and intuitive control of fluid simulations. While the control framework presented here is an important step in this direction, it lacks the possibility to intuitively modify the simulation. This could be achieved by combining the particle based control with techniques from sketch based interfaces [IMT99, KH06], e.g., to allow a user to paint vortices or waves directly into a fluid animation. Such an approach could also be combined with an intelligent restart and break-point algorithm. This would allow the reuse of previously computed unmodified parts of the simulation. In some cases, it might even be interesting to allow direct modifications of the isolevel values of the animation for manual small scale changes [LH06].

## Simulation of Metal Foams

Although the basic free surface algorithm of this thesis was originally developed to simulate metal foams, as described in Section 1, a full simulation of the production

process of metal foams requires the treatment of additional effects [Thi05]. Naturally, the gas of the foaming agent plays an important role. Thus, the fluid simulation has to be combined with an advection diffusion solver to simulate the gas phase, and its effects on the bubble formation [KTH<sup>+</sup>05]. A realistic simulation should furthermore include temperature effects, possibly treating the whole process from melting of the metal pellets to their cooling and solidification. In addition, the foaming processes requires an accurate and efficient surface tension computation, as surface tension plays an important role in the formation and stabilization of the foam structure.

While the algorithms from Sections 5, 9 and 10 are not important for foaming simulations, the adaptive time stepping of Section 6 is interesting to efficiently handle the different time scales that occur during the foam formation. Bubble coalescence, for example, results in momentarily high velocities and surface tension forces in comparison to the overall relatively slow foaming process. The combination of the turbulence model with the free surface algorithm of Section 4 is likewise useful to stabilize a foaming simulation, as the viscosity of, e.g., liquid aluminum is similar to that of water. The adaptive grids of Section 7 might only be useful in the early phases of the foaming process, where the fluid forms a large volume before the individual bubble kernels are formed. Finally, the API described in Section 11 could be used to directly couple the fluid solver to a CAD program, which would allow the accurate setup of the casting form and its initial conditions. Overall, the algorithm is very suitable for metal foam simulations, as the free surface can be represented within a single layer of the computational grid and can handle large density ratios between the two phases. Other techniques that work with a smooth interface layer would require significantly higher grid resolutions to resolve the thin foam structures.

## Particle Technology Applications

Another interesting area of application for the free surface algorithm of this thesis are combined fluid and particle simulations, e.g., for the field of nano-technology engineering. In this case, simulations can be used to increase the understanding of particle dispersions, the agglomeration processes, particle interactions and the particle properties [BFS<sup>+</sup>06, Fei06]. A typical example of a problem from this field is the simulation of charged colloids in a fluid, e.g., for glues and paints [HF01, Fei05]. To study the behavior of such a colloidal dispersion, it would be interesting to simulate the whole process of particle agglomeration on a surface together with the evaporation of the solvent liquid. Other particle processes that are interesting applications for simulation are sedimentation processes, e.g., the sedimentation of sand in a river bed [Igl05].

For these applications, adaptive grids and parallelization (Section 7.8) quickly become important once larger particle numbers should be simulated. While smaller numbers, up to around 10, can be handled by a typical workstation, particle numbers of 1000 and more would be interesting for realistic applications. As each particle has to be represented by a certain number of grid cells (typically a diameter between 6-8 cells), to allow accurate computations of its boundary conditions, the resulting overall resolutions for the fluid volume can be large. The adaptive coarsening is directly applicable to problems with moving particles, to refine the computations of their boundaries, while coarsening the computations of the fluid volumes. Surface tension forces,

however, are often of importance, as they can be large in comparison to the other forces that are involved, e.g., electrostatic and Van-der-Waals forces. Hence, the extensions to compute the surface curvature, as described above, are also important for particle technology applications.

In conclusion, the LBM is a interesting approach for fluid simulations that is further extended by the active LBM research community. The LB free surface algorithm explained in this thesis makes it possible to efficiently perform realistic simulations of free surface fluids for a wide variety of applications.





# Bibliography

- [AA06] Anders Adamson and Marc Alexa. Point-sampled cell complexes. *ACM Trans. Graph.*, 25(3):671–680, 2006.
- [ATKS00] M. Arnold, M. Thies, C. Körner, and R.F. Singer. Experimental and Numerical Investigation of the Formation of Metal Foam. *Material-sweek*, 2000.
- [BdLL01] M'hamed Bouzidi, Dominique d'Humières, Pierre Lallemand, and Li-Shi Luo. Lattice Boltzmann equation on a two-dimensional rectangular grid. *J. Comp. Phys.*, 172(2):704–717, 2001.
- [BFS<sup>+</sup>06] Christian Binder, Christian Feichtinger, Hans-Joachim Schmid, Nils Thürey, Wolfgang Peukert, and Ulrich Rüde. Simulation of the Hydrodynamic Drag of Aggregated Particles. *Journal of Colloid and Interface Science*, 301:155–167, 2006.
- [BGD05] Vivek Buwa, Daniel Gerlach, and Franz Durst. Regimes of bubble formation on submerged orifices. *Phys. Rev. Letters*, April 2005.
- [BGK54] P. L. Bhatnagar, E. P. Gross, and M. Krook. A model for collision processes in gases. *Phys. Rev.*, 94:511–525, 1954.
- [BGOS06] Adam W. Bargteil, Tolga G. Goktekin, James F. O'brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1):19–38, 2006.
- [BR86] J. U. Brackbill and H. M. Ruppel. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys.*, 65(2):314–343, 1986.
- [Bro05] Ilja N. Bronstein. *Taschenbuch der Mathematik*. Harri Deutsch, 2005.
- [BT99] B. Bunner and G. Tryggvason. Direct numerical simulations of three-dimensional bubbly flows. *Physics of Fluids*, 11:1967–1969, August 1999.

- [CCM92] Hudong Chen, Shiyi Chen, and William H. Matthaeus. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method. *Phys. Rev. A*, 45(8):R5339–R5342, 1992.
- [CDK<sup>+</sup>01] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Academic Press, 2001.
- [CKTR03] B. Crouse, M. Krafcyzk, J. Tölke, and E. Rank. A LB-based approach for adaptive flow simulations. *Int. J. Modern Phys. B*, 17:109–112, 2003.
- [CMT04] Mark Carlson, Peter John Mucha, and Greg Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3), 2004.
- [Del01] Paul J. Dellar. Non-hydrodynamic modes and a priori construction of shallow water lattice Boltzmann equations. *Phys. Rev. E*, 65, 2001.
- [dGK<sup>+</sup>02] D. d’Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, , and Li-Shi Luo. Multiple-relaxation-time lattice Boltzmann models in three-dimensions. *Philosophical Transactions of Royal Society of London A*, 360(1792):437–451, 2002.
- [DKvS99] E. Delnoij, J. A. M. Kuipers, and W. P M. van Swaaij. A three-dimensional CFG model for gas-liquid bubble columns. *Chemical Engineering Science*, 54, 1999.
- [Don04] S. Donath. On Optimized Implementations of the Lattice Boltzmann Method on Contemporary Architectures. Bachelor Thesis, High Performance Computing Center, University of Erlangen-Nuremberg, Aug 2004.
- [DS05] P. J. Dellar and R. Salmon. Shallow water equations with a complete Coriolis force and topography. *Phys. Fluids*, 17, 2005.
- [Dur06] Franz Durst. *Grundlagen der Strömungsmechanik*. Springer, 2006.
- [ELF05] D. Enright, F. Losasso, and R. Fedkiw. A Fast and Accurate Semi-Lagrangian Particle Level Set Method. *Computers and Structures*, 83:479–490, 2005.
- [EMF02] D. Enright, S. Marschner, and R. Fedkiw. Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.*, 21(3):736–744, 2002.

- [ETK<sup>+</sup>06] S. Elcott, Y. Tong, E. Kanso, P. Schröder, and M. Desbrun. Stable, Circulation-Preserving, Simplicial Fluids. *to appear, SIGGRAPH 2006 Course Notes: Discrete Differential Geometry*, 2006.
- [FAMO99] Ronald P. Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows. *J. Comp. Phys.*, 152:457–492, 1999.
- [FdH<sup>+</sup>87] Uriel Frisch, Dominique d’Humières, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau, and Jean-Pierre Rivert. Lattice Gas Hydrodynamics in Two and Three Dimensions. *Complex Systems*, 1:649–707, 1987.
- [Fei05] C. Feichtinger. Drag Force Simulations of Particle Agglomerates with the Lattice-Boltzmann Method. Study Thesis, Institute for System Simulation, University of Erlangen-Nuremberg, Jun 2005.
- [Fei06] C. Feichtinger. Simulation of Moving Charged Colloids with the Lattice Boltzmann Method. Diploma Thesis, Institute for System Simulation, University of Erlangen-Nuremberg, Jun 2006.
- [FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proc. of ACM SIGGRPAH*, pages 23–30, 2001.
- [FH98] O. Filippova and D. Hänel. Grid Refinement for Lattice-BGK models. *J. Comp. Phys.*, 147:219–228, 1998.
- [FL04] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Trans. Graph.*, 23(3):441–448, 2004.
- [FM96] N. Foster and D. Metaxas. Realistic Animation of Liquids. *Graphical Models and Image Processing*, 58, 1996.
- [FM97] Nick Foster and Dimitris Metaxas. Controlling fluid animation. In *Proc. of CGI*, 1997.
- [FOA03] Bryan E. Feldman, James F. O’Brien, and Okan Arikan. Animating suspended particle explosions. In *Proc. of ACM SIGGRAPH*, pages 708–715, Aug 2003.
- [FOK05] Bryan E. Feldman, James F. O’Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. *ACM Trans. Graph.*, 24(3):904–909, 2005.

- [Fro79] Arnold Frohn. *Einführung in die kinetische Gastheorie*. Akademische Verlagsgesellschaft Wiesbaden, 1979.
- [GDN88] Michael Griebel, Thomas Dornseifer, and Tilman Neunhöffer. *Numerical Simulation in Fluid Dynamics*. SIAM, 1988.
- [GKT<sup>+</sup>06] S. Geller, M. Krafczyk, J. Tölke, S. Turek, and J. Hron. Benchmark computations based on Lattice-Boltzmann, Finite Element and Finite Volume Methods for laminar Flows. *Computers and Fluids*, 35 [8-9], September-November 2006.
- [GLS99] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. MIT Press, second edition, 1999.
- [GLT99] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2, Advances Features of the Message-Passing Interface*. MIT Press, 1999.
- [GRWS04] Robert Geist, Karl Rasche, James Westall, and Robert Schalkoff. Lattice-Boltzmann Lighting. *Proc. of Eurographics Symposium on Rendering 2004*, pages 355–362, 2004.
- [GRZZ91] A. K. Gunstensen, D. H. Rothman, S. Zaleski, and G. Zanetti. Lattice Boltzmann model of immiscible fluids. *Phys. Rev. A*, 43, 1991.
- [GS03] I. Ginzburg and K. Steiner. Lattice Boltzmann model for Free-Surface flow and its Application to Filling Process in Casting. *J. Comp. Phys.*, 185/1, 2003.
- [GSLF05] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling Water and Smoke to Thin Deformable and Rigid Shells. *ACM Trans. Graph.*, 24(3):973–981, 2005.
- [Gue06] E. Guendelman. Physically-based simulation of solids and solid-fluid coupling. PhD thesis, Stanford University, 2006.
- [Har71] Stewart Harris. *An Introduction to the Theory of the Boltzmann Equation*. Holt, Rinehart and Winston Inc., 1971.
- [HF01] Jürgen Horbach and Daan Frenkel. Lattice-boltzmann method for the simulation of transport phenomena in charged colloids. *Phys. Rev. E*, 64(6), Nov 2001.

- [HHL<sup>+</sup>05] T. R. Hagen, J. M. Hjelmervik, K.-A. Lie, J. R. Natvig, and M. Ofstad Henriksen. Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory*, 13, 2005.
- [HK04] Jeong-Mo Hong and Chang-Hun Kim. Controlling fluid animation with geometric potential: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):147–157, 2004.
- [HK05] Jeong-Mo Hong and Chang-Hun Kim. Discontinuous fluids. *ACM Trans. Graph.*, 24(3):915–920, 2005.
- [HL97a] X. He and L.-S. Luo. A Priori Derivation of Lattice Boltzmann Equation. *Phys. Rev. E*, 55:R6333–R6336, 1997.
- [HL97b] X. He and L.-S. Luo. Lattice Boltzmann model for the incompressible Navier-Stokes equations. *J. Stat. Phys.*, 88:927–944, 1997.
- [HN81] C. W. Hirt and B. D. Nichols. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *J. Comp. Phys.*, 39:201–225, 1981.
- [HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive Animation of Ocean Waves. July 2002.
- [HS66] J. L. Hess and A. M. O. Smith. Calculation of potential flow about arbitrary bodies. *Prog. Aeronaut. Sci.*, 8, 1966.
- [HSCD96] Shuling Hou, James D. Sterling, Shiyi Chen, and Gary Doolen. A Lattice Boltzmann Subgrid Model for High Reynolds Number Flow. *Fields Institute Communications*, 6:151–166, 1996.
- [HYP76] J. Hardy, O. De Pazzis Y., and Pomeau. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Physical Review A*, 13:1949–1960, 1976.
- [Igl03] K. Iglberger. Performance Analysis and Optimization of the Lattice Boltzmann Method in 3D. Study Thesis, Institute for System Simulation, University of Erlangen-Nuremberg, Sep 2003.
- [Igl05] K. Iglberger. Lattice-Boltzmann Simulation of Flow around moving Particles. Master Thesis, Institute for System Simulation, University of Erlangen-Nuremberg, Jun 2005.
- [IGLF06] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques. *ACM Trans. Graph.*, 25, 2006.

- [IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3D freeform design. In *SIGGRAPH '99: Proc. of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999.
- [JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, 2005.
- [KAG<sup>+</sup>05] Richard Keiser, Bart Adams, Dominique Gasser, Paolo Bazzi, Philip Dutre, and Markus Gross. A Unified Lagrangian Approach to Solid-Fluid Animation. *Proc. of the 2005 Eurographics Symposium on Point-Based Graphics*, 2005.
- [KAG<sup>+</sup>06] Richard Keiser, Bart Adams, Leonidas J. Guibas, Philip Dutre, and Mark Pauly. Multiresolution particle-based fluids . CS Technical Report 520. Technical report, Computer Science Department, ETH Zurich, 2006.
- [KAK<sup>+</sup>06] Vivek Kwatra, David Adalsteinsson, Theodore Kim, Nipun Kwatra, Mark Carlson, and Ming Lin. Texturing Fluids. *SIGGRAPH Technical Sketch*, 2006.
- [KBA<sup>+</sup>00] C. Körner, F. Berger, M. Arnold, C. Stadelmann, and R.F. Singer. Influence of Processing Conditions on Morphology of Metal Foams Produced from Metal Powder. *Materials Science and Technology*, 16:781–784, 2000.
- [KC04] Pijush Kundu and Ira Cohen. *Fluid Mechanics*. Elsevier Academic Press, 2004.
- [KCC<sup>+</sup>06] J. Kim, D. Cha, B. Chang, B. Koo, and I. Ihm. Practical Animation of Turbulent Splashing Water. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.
- [KFCO06] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. *ACM Trans. Graph.*, 25(3):820–825, 2006.
- [KH06] Olga A. Karpenko and John F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

- [Kir04] Andrew Kirmse. *Game Programming Gems 4*. Charles River Media, 2004.
- [KKKA05] Ryo Kondo, Takashi Kanai, and Ken-ichi Anjyo. Directable animation of elastic objects. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 2005.
- [KM90] M. Kass and G. Miller. Rapid, Stable Fluid Dynamics for Computer Graphics. *ACM Trans. Graph.*, 24(4):49–55, 1990.
- [KPR<sup>+</sup>05] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Zeiser. Parallel Lattice Boltzmann Methods for CFD Applications. In A.M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *LNCSE*, pages 439–465. Springer, 2005.
- [Kra01] Manfred Krafczyk. Gitter-Boltzmann-Methoden - von der Theorie zur Anwendung. Habilitation, 2001.
- [KS99] C. Körner and R.F. Singer. Numerical Simulation of Foam Formation and Evolution with Modified Cellular Automata. *Metal Foams and Porous Metal Structures*, pages 91–96, 1999.
- [KS00] C. Körner and R.F. Singer. Processing of Metal Foams - Challenges and Opportunities. *Advanced Engineering Materials*, 2(4):159–65, 2000.
- [KTH<sup>+</sup>05] C. Körner, M. Thies, T. Hofmann, N. Thürey, and U. Rüde. Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming. *Journal of Statistical Physics*, 121 [1-2]:179–196, October 2005.
- [KTL03] M. Krafczyk, J. Tlke, and Li-Shi Luo. Large-eddy simulations with a multiple-relaxation-time LBE model. *International Journal of Modern Physics B*, 17:33–39, 2003.
- [KTS02] C. Körner, M. Thies, and R. F. Singer. Modeling of Metal Foaming with Lattice Boltzmann Automata. *Advanced Engineering Materials*, 2002.
- [Lad94] A. J. C. Ladd. Numerical simulation of particular suspensions via a discretized Boltzmann equation. Part 2. Numerical results. *J. Fluid Mech.*, 39:271–311, 1994.

- [lBefitpfwldd04] A lattice Boltzmann method for incompressible two-phase flows with large density differences. T. Inamuro and T. Ogata and S. Tajima and N. Konishi. *J. Comp. Phys.*, 198:628–644, 2004.
- [LC87] William Lorensen and Harvey Cline. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. In *Computer Graphics Vol. 21, No. 4*, pages 163–169, August 1987.
- [LF02] Arnauld Lamorlette and Nick Foster. Structural modeling of flames for a production environment. In *Proc. of ACM SIGGRAPH*, pages 729–735, 2002.
- [LGF04] F. Losasso, F. Gibou, and R. Fedkiw. Simulating Water and Smoke With an Octree Data Structure. *ACM Trans. Graph.*, 23(3):457–462, 2004.
- [LH06] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. *ACM Trans. Graph.*, 25(3):579–588, 2006.
- [LIG06] Frank Losasso, Geoffrey Irving, and Eran Guendelman. Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):343–352, 2006. Member-Ron Fedkiw.
- [LL00] Pierre Lallemand and Li-Shi Luo. Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E*, 61(6):6546–6562, 2000.
- [LLS00] D. P. Lockard, L.-S. Luo, and B. A. Singer. Evaluation of the lattice-Boltzmann equation solver PowerFLOW for aerodynamic applications. Technical report, ICASE, 2000.
- [Lov03] Jörn Loviscach. Complex Water Effects at Interactive Frame Rates. *Journal of WSCG*, 11:298–305, 2003.
- [LSSF06] Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819, 2006.
- [LWK03] Wei Li, Xiaoming Wei, and Arie E. Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, 2003.
- [MC98] A. Masselot and B. Chopard. A lattice boltzmann model for particle transport and deposition. *Europhys. Lett.*, 42:259–264, 1998.

- [MCG03a] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 154–159, 2003.
- [MCG03b] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 2003.
- [MKN<sup>+</sup>04] Matthias Müller, Richard Keiser, Andrew Nealen, Mark Pauly, Markus Gross, and Marc Alexa. Point based animation of elastic, plastic and melting objects. In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, Aug 2004.
- [MMS04] V. Mihalef, D. Metaxas, and M. Sussman. Animation and Control of Breaking Waves. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 315–324, 2004.
- [Mon05] J. J. Monaghan. Smoothed particle hydrodynamics. *Rep. Prog. Phys.*, 68:1703–1758, 2005.
- [MSRG05] Matthias Müller, Barbara Solenthaler, Richard, and Markus Gross. Particle-based fluid-fluid interaction. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 2005.
- [MST<sup>+</sup>04] Matthias Müller, Simon Schirm, Matthias Teschner, Bruno Heidelberger, and Markus Gross. Interaction of Fluids with Deformable Solids. *Journal of Computer Animation and Virtual Worlds (CAVW)*, 15(3-4):159–171, July 2004.
- [MTPS04] Antoine McNamara, Adrien Treuille, Zoran Popovic, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, 2004.
- [MZ88] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Phys. Rev. Lett.*, 61(20):2332–2335, 1988.
- [N. 06] N. Thürey and T. Pohl and U. Rüde and M. Oechsner and C. Körner. Optimization and Stabilization of LBM Free Surface Flow Simulations using Adaptive Parameterization. *Computers and Fluids*, 35 [8-9]:934–939, September-November 2006.

- [NFJ02] Duc Quang Nguyen, Ronald Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 721–728, New York, NY, USA, 2002. ACM Press.
- [PCS04] Frederic Pighin, Jonathan M. Cohen, and Maurya Shah. Modeling and editing flows using advected radial basis functions. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 223–232. ACM Press, 2004.
- [PK97] H. R. Pruppacher and J. D. Klett. *Microphysics of Clouds and Precipitation*. Springer, 1997.
- [PKA<sup>+</sup>05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutre, Markus Gross, and Leonidas J. Guibas. Meshless Animation of Fracturing Solids. *ACM Trans. Graph.*, 24:957–964, 2005.
- [PKKG03] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.
- [PKW<sup>+</sup>03] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rüde. Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes in 2D and 3D. Technical Report 03–8, Germany, 2003.
- [Poh07] Thomas Pohl. Free-Surface Flow Simulation Using the Lattice Boltzmann Method on High Performance Computers. PhD Thesis, Institute for System-Simulation, University of Erlangen-Nuremberg, 2007.
- [PP04] James Edward Pilliod and Elbridge Gerry Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *J. Comput. Phys.*, 199(2):465–502, 2004.
- [PPG04] Mark Pauly, Dinesh Pai, and Leo Guibas. Quasi-Rigid Objects in Contact. *Proc. of ACM Siggraph/Eurographics Symposium on Computer Animation*, 2004.
- [PTD<sup>+</sup>04] T. Pohl, N. Thürey, F. Deserno, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser. Performance Evaluation of Parallel Large-Scale Lattice

- Boltzmann Applications on Three Supercomputing Architectures. In *Proc. of Supercomputing Conference 2004*, 2004.
- [QdL92] Y. H. Qian, D. d’Humières, and P. Lallemand. Lattice BGK Models for Navier-Stokes Equation. *Europhys. Lett.*, 17(6):479–484, 1992.
- [REN<sup>+</sup>04] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 2004.
- [RNGF03] Nick Rasmussen, Duc Quang Nguyen, Willi Geiger, and Ronald Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph.*, 22(3):703–707, 2003.
- [Roh04] M. Rohde. Extending the lattice-Boltzmann method: Novel techniques for local grid refinement and boundary conditions. PhD Thesis, Technische Universiteit Delft, 2004.
- [SC94] X. Shan and H. Chen. Simulation of non-ideal gases and liquid-gas phase transitions by the lattice Boltzmann equation. *Phys. Rev. E*, 49:2941–2948, 1994.
- [Sma63] J. Smagorinsky. General circulation experiments with the primitive equations. *Mon. Wea. Rev.*, 91:99–164, 1963.
- [SOOY96] M. R. Swift, E. Orlandi, W. R. Osborn, and J. M. Yeomans. Lattice Boltzmann simulations of liquid-gas and binary fluid systems. *Phys. Rev. E*, 54:5041–5052, 1996.
- [SRF05] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, 2005.
- [Sta99] Jos Stam. Stable Fluids. *Proc. of ACM SIGGRAPH*, pages 121–128, 1999.
- [Sta03] Jos Stam. Real-Time Fluid Dynamics for Games. *Proc. of the Game Developer Conference*, March 2003.
- [SU94] J. Steinhoff and D. Underhill. Modification of the Euler Equations for Vorticity Confinement: Application to the Computation of Interacting Vortex Rings. volume 6, 8, pages 2738–2744, 1994.

- [Suc01] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.
- [Sus03] M. Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comp. Phys.*, 187/1, 2003.
- [SY05a] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Trans. Graph.*, 24(1), 2005.
- [SY05b] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 2005.
- [SZ99] R. Scardovelli and S. Zaleski. Direct numerical simulation of free-surface and interfacial flow. *Annu. Rev. Fluid Mech.*, 31:567–603, 1999.
- [SZ04] H. Struchtrup and Y. Zheng. Burnett equations for the ellipsoidal statistical bgk model. *Cont. Mech. Thermodyn.*, 16(1-2):97–108, 2004.
- [Tes04] Jerry Tessendorf. Simulating Ocean Surfaces. *SIGGRAPH 2004 Course Notes 31*, 2004.
- [TFK<sup>+</sup>03a] T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki. Realistic Animation of Fluid with Splash and Foam. *Computer Graphics Forum*, 22 (3), 2003.
- [TFK03b] J. Tölke, S. Freudiger, and M. Krafczyk. An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations. *Computers & Fluids*, 17:109–112, 2003.
- [Thi05] M. Thies. Lattice Boltzmann Modeling with Free Surfaces Applied to Formation of Metal Foams. PhD Thesis, Institute for Material Science, University of Erlangen-Nuremberg, 2005.
- [Thu03] Nils Thuerey. A Lattice Boltzmann method for single-phase free surface flows in 3D. Masters thesis, Dept. of Computer Science 10 System-Simulation, University of Erlangen-Nuremberg, 2003.
- [Thu06] Nils Thuerey. Fluid Simulation with Blender. *Dr. Dobbs Journal*, January 2006.
- [TKPR06] N. Thürey, Richard Keiser, Mark Pauly, and U. Rüde. Detail-Preserving Fluid Control. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.

- [TKSR02] J. Tölke, M. Krafcyzk, M. Schulz, and E. Rank. Lattice Boltzmann Simulations of binary fluid flow through porous media. *Phil. Trans. R. Soc. Lond. A*, 360:535–545, 2002.
- [TLP06] Adrien Treuille, Andrew Lewis, and Zoran Popovic. Model reduction for real-time fluids. *ACM Trans. Graph.*, 25(3):826–834, 2006.
- [TMPS03] Adrien Treuille, Antoine McNamara, Zoran Popovic, and Jos Stam. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22(3):716–723, 2003.
- [TOS01] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.
- [TR04] N. Thürey and U. Rüde. Free Surface Lattice-Boltzmann fluid simulations with and without level sets. *Proc. of Vision, Modelling, and Visualization VMV*, pages 199–208, 2004.
- [TR06] N. Thürey and U. Rüde. Stable Free Surface Flows with the Lattice Boltzmann Method on adaptively coarsened Grids. *submitted to Computing and Visualization in Science, preprint version available online*, 2006.
- [Tre02] Jan Treibig. Simulation von Gas-Feststoff-Mehrphasensystemen mit dem Lattice Boltzmann Verfahren. Diploma Thesis, University of Erlangen-Nuremberg, 2002.
- [TRS06] N. Thürey, U. Rüde, and Marc Stamminger. Animation of Open Water Phenomena with coupled Shallow Water and Free Surface Simulations. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.
- [Wel54] P. Welander. On the temperature jump in a rarefied gas. *Arkiv Fysik*, 7, 1954.
- [WG00] Dieter A. Wolf-Gladrow. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*. Springer, 2000.
- [Wil03] J. Wilke. Cache Optimizations for the Lattice Boltzmann Method in 2D. Study Thesis, Institute for System Simulation, University of Erlangen-Nuremberg, Feb 2003.
- [WLMK04] Xiaoming Wei, Wei Li, Klaus Müller, and Arie E. Kaufman. The Lattice-Boltzmann Method for Simulating Gaseous Phenomena.

- IEEE Transactions on Visualization and Computer Graphics*, 10(2):164–176, 2004.
- [WMT05] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005.
- [WMT06] Chris Wojtan, Peter J. Mucha, and Greg Turk. Control of Complex Particle Systems Using the Adjoint Method. *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006.
- [Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.
- [WWXP06] Changbo Wang, Zhangye Wang, Tian Xia, and Qunsheng Peng. Real-time snowing simulation. *The Visual Computer*, pages 315–323, May 2006.
- [WZF<sup>+</sup>03] Xiaoming Wei, Ye Zhao, Zhe Fan, Wei Li, Suzanne Yoakum-Stover, and Arie Kaufman. Natural phenomena: Blowing in the wind. *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 75–85, July 2003.
- [YGL05a] H. Yu, S.S. Girimaji, and Li-Shi Luo. Lattice Boltzmann simulations of decaying homogeneous isotropic turbulence. *Phys. Rev. E*, 71, 2005.
- [YGL05b] Huidan Yu, Sharath S. Girimaji, and Li-Shi Luo. DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method. *J. Comput. Phys.*, 209(2):599–616, 2005.
- [YLG06] H. Yu, L.-S. Luo, and S. S. Girimaji. LES of turbulent square jet flow using an MRT lattice Boltzmann model. *Computers & Fluids*, 25:957–965, 2006.
- [YMLS03] Dazhi Yu, Renwei Mei, Li-Shi Luo, and Wei Shyy. Viscous flow computations with the method of lattice Boltzmann equation. *Progress in Aerospace Sciences* 39, 5, 2003.
- [YMS02] Dazhi Yu, Renwei Mei, and Wei Shyy. A multi-block lattice Boltzmann method for viscous fluid flows. *Int. J. for Numerical Methods in Fluids*, 39, 2002.
- [YUM86] Larry Yaeger, Craig Upson, and Robert Myers. Combining physical and visual simulation and creation of the planet jupiter for the film 2010. In *SIGGRAPH '86: Proceedings of the 13th annual conference on*

- Computer graphics and interactive techniques*, pages 85–93. ACM Press, 1986.
- [ZB05] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.
- [Zho04] Jian Guo Zhou. *Lattice Boltzmann Methods for Shallow Water Flows*. Springer, 2004.



# Appendix A

## German Parts

### A.1 Inhaltsverzeichnis

Titel: *Physikalische Animation von Strömungen mit freien Oberflächen mit der Lattice-Boltzmann-Methode*

<b>1 Einleitung</b>	<b>1</b>
<b>2 Strömungen mit freien Oberflächen</b>	<b>5</b>
2.1 Animation von freien Oberflächen . . . . .	5
2.2 Gegenüberstellung von Simulations-Ansätzen . . . . .	7
<b>3 Die Lattice-Boltzmann Methode</b>	<b>11</b>
3.1 Historische Entwicklung . . . . .	11
3.2 Der Basis Algorithmus . . . . .	12
3.3 Stabilität . . . . .	15
3.4 Parametrisierung . . . . .	17
3.5 Herleitung . . . . .	18
3.5.1 Die Navier-Stokes Gleichungen . . . . .	18
3.5.2 Die Boltzmann Gleichungen . . . . .	19
3.5.3 Chapman-Enskog Entwicklung . . . . .	20
3.5.4 Herleitung der Lattice Boltzmann Gleichungen . . . . .	21
3.6 Abschluss . . . . .	25
<b>4 Simulation von Strömungen mit freien Oberflächen</b>	<b>27</b>
4.1 Bewegung der Grenzfläche . . . . .	29
4.2 Randbedingungen . . . . .	30
4.3 Initialisierung der Zelltypen . . . . .	31
4.4 Vermeidung von Artefakten . . . . .	33
4.5 Interaktive Simulationen . . . . .	33
<b>5 Bewegliche Hindernisse</b>	<b>37</b>
5.1 Randbedingungen für Hindernisse . . . . .	38
5.2 Randbedingungen für bewegliche Ränder . . . . .	38
5.3 Initialisierung des Gitters . . . . .	39

5.4	Generierung der Oberflächen . . . . .	42
5.5	Ergebnisse . . . . .	44
<b>6</b>	<b>Adaptive Zeitschritte</b>	<b>47</b>
6.1	Adaptive Parametrisierung . . . . .	47
6.2	Validierung und Performanz . . . . .	50
<b>7</b>	<b>Adaptive Gitter</b>	<b>53</b>
7.1	Gitter Verfeinerung . . . . .	53
7.2	Ein Algorithmus zur adaptiven Vergrößerung . . . . .	55
7.3	Validierung . . . . .	61
7.4	Performanz . . . . .	64
<b>8</b>	<b>Parallelisierung</b>	<b>71</b>
8.1	Parallelisierung mit OpenMP . . . . .	71
8.2	Parallelisierung mit MPT . . . . .	75
<b>9</b>	<b>Fluid Kontrolle</b>	<b>79</b>
9.1	Verwandte Arbeiten . . . . .	79
9.2	Erzeugung von Kontrollpartikeln . . . . .	82
9.3	Kontrollkräfte . . . . .	82
9.4	Detail-erhaltende Kontrolle . . . . .	84
9.5	Ergebnisse . . . . .	85
<b>10</b>	<b>Modellierung von Fluiden auf grossen Skalen</b>	<b>89</b>
10.1	Die Flachwassergleichungen . . . . .	90
10.2	Hybride 2D/3D Simulation . . . . .	92
10.3	Ein Modell für Tropfen . . . . .	96
10.4	Ergebnisse und Diskussion . . . . .	99
<b>11</b>	<b>Anbindung von 3D Anwendungen</b>	<b>101</b>
11.1	Integration in Blender . . . . .	101
11.2	Erweiterungen zur Integration . . . . .	102
<b>12</b>	<b>Abschluss</b>	<b>107</b>
12.1	Zusammenfassung . . . . .	107
12.2	Diskussion and zukünftige Arbeiten . . . . .	108
<b>A</b>	<b>Teile der Arbeit auf Deutsch</b>	<b>127</b>
<b>B</b>	<b>Lebenslauf</b>	<b>131</b>

## A.2 Zusammenfassung

Numerische Strömungssimulationen sind zu einem wichtigen Hilfsmittel vieler technischer Anwendungen geworden. Dabei stellen besonders die Strömungen mit freien Oberflächen einen Spezialfall dar, der in vielen Bereichen von Bedeutung ist. Dieser

Arbeit konzentriert sich auf die Simulation eines solches System mit zwei Phasen, z.B. Wasser und Luft. Die freie Oberfläche wird dabei mittels einer einzelnen Fluidphase und einer scharfen Grenzfläche mit geeigneten Randbedingungen simuliert. Fokus dieser Arbeit ist die Erstellung von Animationen freier Oberflächen. Weitere Anwendungen sind jedoch unter anderem (Rechtschreibung checken) die Simulation von Metallschäumen und von Partikeln (oder Partikelsystemen) in einer Strömung.

Der in dieser Arbeit verwendete Simulationsalgorithmus basiert auf der Lattice-Boltzmann Methode. Dieser Simulationsansatz ermöglicht die Simulation von komplexen Geometrien und Topologien mit hoher Effizienz. Der Basisalgorithmus wird im Folgenden um die masseerhaltende Behandlung freier Oberflächen erweitert. Adaptive Zeitschritte und adaptive Gitter in Kombination mit einem LES- Modell ermöglichen die effiziente Simulation detaillierter Strömungen. Im Zusammenspiel mit Randbedingungen für bewegliche und verformbare Objekte stellt dies eine flexible Basis für Simulationen von Fluiden mit freien Oberflächen dar.

Für Fluidsimulationen in der Filmindustrie ist es jedoch zusätzlich wichtig, die Simulation zu kontrollieren. Hierzu wird ein Ansatz vorgestellt, der eine effiziente Kontrolle ermöglicht, die die natürliche Bewegung des Fluids jedoch so wenig wie möglich verändert. Bei der Simulation von grossflächigen Wasseroberflächen stellen die unterschiedlichen Skalen von Wellen bis hin zu Tropfen ein technisches Problem dar. Um solche Simulationen durchzuführen, wird eine Kombination von zwei- und dreidimensionalen Techniken vorgestellt. Um die Fähigkeiten des im Rahmen dieser Arbeit angefertigten Lösers zu demonstrieren, wurde er in die Open Source Anwendung Blender integriert.

Am Ende der Arbeit werden weitere Bereiche für zukünftige Arbeiten und mögliche Erweiterungen des Algorithmus vorgestellt. Dabei ist z.B. die genaue und schnelle Berechnung der Oberflächenspannung ein wichtiges Thema. Des Weiteren werden Erweiterungen für die Simulation von Metallschäumen und von Partikelsystemen diskutiert.

## A.3 Einleitung

Materialien mit einem flüssigen Aggregatzustand spielen in einer Vielzahl von natürlichen Begebenheiten eine essentielle Rolle. Dadurch sind sie sowohl im Alltag als auch in vielen Produktionsprozessen wichtig. Die numerische Simulation von gasförmigen und flüssigen Stoffen ist ein wichtiges Hilsmittel geworden, um z.B. eine Wettervorhersage zu machen oder das optimale Profil einer Tragfläche zu bestimmen. In dieser Arbeit geht es vor allem um Probleme mit zwei Phasen, bei denen eine davon mit einem vereinfachten Modell behandelt werden kann. Dieser Ansatz ist als Behandlung von freien Oberflächen bekannt. In Bild 1.1 und 1.2 sind verschiedene Beispiele von Strömungen mit freien Oberflächen zu sehen, die von einem solchen Simulator reproduziert werden können. Obwohl freie Oberflächen einen Spezialfall darstellen, sind sie dennoch für viele verschiedene Anwendungen, wie Gussprozesse oder die Erstellung von Animationen von Bedeutung.

Funktionen für physikalisch basierte Animationen sind mittlerweile Bestandteil vieler grosser 3D-Anwendungspakete, da physikalische Effekte nur schlecht mit kon-

---

ventionellen Methoden animiert werden können. Bei physikalisch basierten Methoden werden die aus der Physik bekannten Gleichungen gelöst oder angenähert um realistische und interessante Bewegungen zu erstellen. Typische Beispiele sind zusammstürzenden Kistenstapel oder die Bewegung von Kleidung von animierten Charakteren. Algorithmen für physikalisch basierte Animationen stellen oft eine Kombination von Algorithmen aus ingenieurwissenschaftlichen Anwendungen und der Computergrafik dar. Während die Ingenieurwissenschaften vor allem an der Genauigkeit einer Berechnung interessiert sind, gibt es ein solches Mass bei Animationen nicht, da hier vor allem der visuelle Eindruck eine Rolle spielt. Für praktische Anwendungen ist die Effizienz der Algorithmen jedoch sehr wichtig. Weiterhin muss die Massenerhaltung garantiert sein und die sehr niedrige Viskosität von typischen Flüssigkeiten wie Wasser muss vom Algorithmus behandelt werden können.

Die physikalisch basierte Animation von Flüssigkeiten stützt sich auf die bekannten Navier-Stokes Gleichungen, die schon seit langem erfolgreich numerisch gelöst werden. Die ersten zweidimensionalen Umströmungen eines Zylinders wurden schon 1930 berechnet, während es bis ca. 1965 dauerte, bis erste dreidimensionale Probleme gelöst wurden. In dieser Arbeit wird ein Verfahren, dass zur Simulation von Metallschäumen entwickelt wurde, erweitert um die Erstellung von komplexen dreidimensionalen Animationen von freien Oberflächen zu ermöglichen. Der Algorithmus baut auf der *Lattice-Boltzmann* Methode (LBM) auf, die einen relativ neuen Ansatz zur Lösung der Navier-Stokes Gleichungen darstellt. Die zunehmende Popularität ist dabei vor allem auf den einfachen Basis-Algorithmus und die hohe Effizienz zurückzuführen.

## Beiträge dieser Arbeit

Das Ziel dieser Arbeit ist die effiziente, stabile und flexible Simulation von Strömungen mit freien Oberflächen mit der LBM. Um dies zu erreichen liefert diese Arbeit unter anderem folgende Beiträge:

- Einen Algorithmus zur Behandlung von beweglichen und verformbaren Hindernissen, um dynamische und realistische Szenen zu ermöglichen.
- Adaptive Zeitschritte für stabile und effiziente Simulationen mit variierenden Geschwindigkeiten.
- Adaptive Gitter zur Beschleunigung von Berechnungen grosser Fluidvolumen. Dieser Ansatz kann die Rechenzeit um einen Faktor von mehr als drei verringern.
- Einen Kontrollmechanismus, der die natürliche Bewegung des Fluids so wenig wie möglich stört.
- Einen hybriden Ansatz zur Simulation von grossflächigen Wasseroberflächen mit kombinierten zwei- und dreidimensionalen Techniken. In Verbund mit einem partikelbasierten Tropfenmodell kann der Eindruck großer Skalen noch verbessert werden.

# **Appendix B**

# **Curriculum Vitae**

## **General Information**

Date / Place of Birth: 1979-07-06, Braunschweig, Germany  
Office Address: Cauerstr. 6, Room 0.146  
D-91058 Erlangen, Germany  
Telephone: +49 (0)9131 85-28691  
Email: [nils@thuerey.de](mailto:nils@thuerey.de)  
WWW: <http://www.ntoken.com>

## **Education**

- Jan. 2004 – Oct. 2006 PhD Student at the Institute for System Simulation,  
University of Erlangen-Nuremberg, Germany
- Feb. 2005 – Apr. 2005 Research visit at the Applied Geometry Group,  
ETH Zurich, Switzerland
- Aug. 2003 – Dec. 2003 Research visit at the Lawrence-Livermore National Laboratory,  
California, USA
- Oct. 1998 – Jun. 2003 Study of Computer Science at the  
University of Erlangen-Nuremberg
- Sept. 1985 – Jun. 1998 Graduation (*Abitur*) at the German international School  
the Hague, Netherlands

## Reviewed Publications

- SCA '06                    *Detail-Preserving Fluid Control*  
                                N. Thürey, R. Keiser, M. Pauly, U. Rüde
- SCA '06                    *Animation of Open Water Phenomena with coupled Shallow Water and Free Surface Simulations*  
                                N. Thürey, U. Rüde, M. Stamminger
- Computers & Mathematics with Applications                    *Simulation of Moving Particles in 3D with the Lattice Boltzmann Method*  
                                K. Iglberger, N. Thürey, U. Rüde
- Comp. Vis. in Science (submitted)                    *Stable Free Surface Flows with the Lattice Boltzmann Method on adaptively coarsened Grids*  
                                N. Thürey, U. Rüde
- VMV '06                    *Free Surface Flows with Moving and Deforming Objects*  
                                N. Thürey, K. Iglberger, U. Rüde
- VMV '06                    *Enhanced Motion Blur Calculation with Optical Flow*  
                                Z. Yuanhang, H. Köstler, N. Thürey, U. Rüde
- J. Coll. and If. Science                    *Simulation of Hydrodynamic Drag of Aggregated Particles*  
                                C. Binder, C. Feichtinger, H.-J. Schmid, N. Thürey,  
                                W. Peukert, U. Rüde
- Lecture Notes in CSE                    *Parallel LBM for CFD Applications*  
                                C. Körner, T. Pohl, U. Rüde, N. Thürey and T. Zeiser  
                                Book chapter in *Numerical Solution of Partial Differential Equations on Parallel Computers*
- Computers and Fluids                    *Optimization and Stabilization of LBM Free Surface Flow Simulations using Adaptive Parameterization*  
                                N. Thürey, T. Pohl, U. Rüde, M. Öchsner, C. Körner
- J. Stat. Phys.                    *Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming*  
                                C. Körner, M. Thies, T. Hofmann, N. Thürey, U. Rüde
- Supercomputing '04                    *Performance Evaluation of Parallel Large-Scale LBM Applications on 3 Supercomputing Architectures*  
                                T. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers,  
                                G. Wellein, T. Zeiser

VMV '04    *Free Surface Lattice-Boltzmann fluid simulations with and without level sets*  
N. Thürey, U. Rüde

PARA '02    *Performance Optimization of 3D Multigrid on Hierarchical Memory Architectures*  
M. Kowarschik, U. Rüde, Nils Thürey, Christian Weiss

## Other Publications

Technical Report 05-4    *Interactive Free Surface Fluids with the Lattice Boltzmann Method*  
N. Thürey, C. Körner, U. Rüde

Dr. Dobbs Journal    *Fluid Simulation with Blender*, N. Thürey

ASIM '05    *Simulation of moving Nano-Particles with the Lattice Boltzmann Method in 3D*  
K. Iglberger, N. Thürey, U. Rüde, H.-J. Schmid, W. Peukert

ASIM '05    *Drag Force Simulations of Particle Agglomerates with the Lattice-Boltzmann Method*  
C. Feichtinger, N. Thürey, U. Rüde, C. Binder, H.-J. Schmid, W. Peukert

Master Thesis    *A single-phase free-surface Lattice-Boltzmann Method*  
University of Erlangen-Nuremberg, 2003

Study Thesis    *Cache Optimizations for Multigrid in 3D*  
University of Erlangen-Nuremberg, 2002

---