

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/227063618>

Stable free surface flows with the Lattice Boltzmann method on adaptively coarsened grids

Article in *Computing and Visualization in Science* · June 2009

DOI: 10.1007/s00791-008-0090-4

CITATIONS

75

READS

823

2 authors, including:



[Ulrich Rüde](#)

Friedrich-Alexander-University of Erlangen-Nürnberg

520 PUBLICATIONS 6,353 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Project GESOP [View project](#)



Physikalisches Management störender Schäume [View project](#)

Lehrstuhl für Informatik 10 (Systemsimulation)



**Technical Report on Turbulent Free Surface Flows
with the Lattice Boltzmann Method on adaptively coarsened
Grids**

Nils Thürey, Ulrich Rüde

Technical Report 05-7

Turbulent Free Surface Flows with the Lattice Boltzmann Method on adaptively coarsened Grids

Nils Thürey , Ulrich Rüde ¹

*University of Erlangen-Nuremberg, Computer Science 10 - System Simulation (LSS),
Cauerstr. 6, 91058 Erlangen, Germany*

Abstract

In this paper we will present an algorithm to simulate free surface flows based on the lattice Boltzmann method (LBM). It makes use of adaptive grids to reduce the required computational time by more than a factor of three for simulations with large volumes of fluid. To achieve this, we combine the free surface treatment with a Smagorinski turbulence model and a technique for adaptive time steps. The simulation inside of large fluid regions is then performed with a coarser resolution. We have developed a set of rules to dynamically adapt the coarse regions to the movement of the free surface, while ensuring the consistency of all grids. The efficiency is partly gained due to simplified interpolation at the grid transfer boundaries. The method is validated by comparing the position of the free surface with an uncoarsened simulation. We will show that the adaptivity in combination with the interpolation causes only a slight decrease of accuracy. The performance is measured with a falling drop and a breaking dam test case for resolutions of up to 480^3 . For these cases our algorithm yields speedups of up to 3.85 for a simulation with three coarser grid levels.

Math. Rev. categories: 65C20, 76M25, 76T05

Key words: lattice Boltzmann methods, free surface fluid simulations, adaptive grids, adaptive time steps, large eddy simulation

PACS: 02.60.Cb, 47.11.-j, 47.55.Ca, 47.27.ep

1 Introduction

Free surface flows are important for a variety of applications, such as the optimization of production processes for foaming or casting [15], the research of bubble

¹ Corresponding author. E-mail address: Nils.Thuerey@cs.fau.de, Phone: +49 9131 85-28691, Fax: +49 9131 85-28928



Figure 1. Two examples of free surface fluid simulations created with the method described in this paper.

formation regimes [2, 8] or for applications in civil engineering such as fluid structure interactions [17]. It is furthermore of importance for physically based animations in computer graphics, as a realistic fluid is hard to achieve without relying on the equations that govern its motion [23]. However, for all simulation problems that appear in these cases it is still problematic to ensure stability and reasonable computation times for turbulent flows.

Our fluid simulation uses the lattice Boltzmann method which can efficiently handle irregular fluid geometries and topologies [7, 20]. Other approaches for free surface simulations make use of level sets in combination with a conventional Navier-Stokes solver [4], or used smoothed particle hydrodynamics to capture the motion of the free surface [18]. The algorithm that we will present in this paper is based on the free surface algorithm that was developed to simulate metal foams [14, 25]. The approach is similar to volume-of-fluid methods, that are often used in cases where mass conservation has to be guaranteed [11]. It furthermore does not require a simulation of the gas phase, and thus saves significant amounts of work for cases with large gas regions [13].

We will first give an overview of the basic algorithm and its extensions. This will include the free surface boundary treatment, a Smagorinski turbulence model, a method to resize the time step and the standard approach to LBM simulations on multiple grids. Afterwards we will discuss how to combine these extensions and present our adaptive coarsening algorithm. The goal of our approach is to efficiently compute the motion of the free surface. Thus, our criterion for coarsening is given by the distance to the free surface. In Section 4, the accuracy of our method will be validated with an error metric that measures the difference of two free surface positions. Afterwards we will present performance measurements for two different simulation setups with varying grid resolutions.

2 The Lattice Boltzmann Method

For the implementation of this paper we have chosen the *lattice Boltzmann method* (LBM), which was derived from the lattice gas methods and can be regarded as a first order explicit discretization of the Boltzmann equation discretized in phase space. Currently there are two different ways of showing that this discretization approximates the *Navier-Stokes* (NS) equations – either by the method of Chapman-

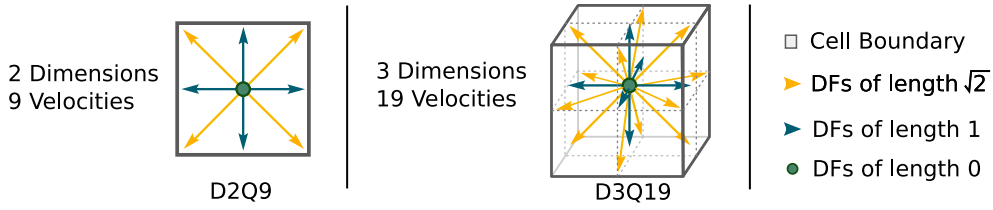


Figure 2. The most commonly used LBM models in two and three dimensions.

Enskog expansion from statistical physics [6], or by direct discretization of the Boltzmann equation [9]. A more detailed overview of the basic algorithm together with extensions and applications can be found e.g. in [24].

For the LBM the velocity space of the molecules or particles in the fluid is discretized. Hence, depending on the dimension and the number of velocity directions, there are different models that can be used. We apply the *D3Q19* model with 19 velocity vectors in three dimensions, as it was shown to have good numerical properties. For two dimensions, the *D2Q9* model with nine velocities is commonly used. For clarity the following illustrations will be based on this model, while the simulations themselves are performed in three dimensions. The velocity vectors $\mathbf{e}_1, \dots, \mathbf{e}_{19}$ of the *D3Q19* model are shown in Fig. 2. For each velocity vector a particle distribution function (DF) is stored. A DF f_i represents an amount of fluid moving with the velocity \mathbf{e}_i . In the following, a DF with subscript \tilde{i} will denote the value from the reverse direction of a DF with subscript i , thus $\mathbf{e}_{\tilde{i}} = -\mathbf{e}_i$.

For simplicity, the size of a cell Δx and the length of a time step Δt both are normalized to a dimensionless 1. The normalization procedure is explained below in more detail. Thus in the *D3Q19* model there are particles not moving at all (f_1), moving with speed 1 (f_2, \dots, f_7) and moving with speed $\sqrt{2}$ (f_8, \dots, f_{19}).

The basic LBM consists of two steps, the stream- and the collide-step. An overview over the two steps of the algorithm is given in Fig. 3. Here the streaming step represents the advection of the particles in the fluid. Streamed DFs f'_i thus can be written as:

$$f'_i(\mathbf{x}, t + \Delta t) = f_{\tilde{i}}(\mathbf{x} + \mathbf{e}_i, t). \quad (1)$$

Due to the normalization of Δx and Δt this results in copying each DF to its adjacent cell along the corresponding velocity vector. The particle collisions that take place during the movement of the particles in the fluid are represented by relaxing the streamed DFs of a cell with density ρ and fluid velocity \mathbf{u} towards the equilibrium distribution function:

$$f_i^{eq} = w_i \left[\rho + 3\mathbf{e}_i \cdot \mathbf{u} - \frac{3}{2}\mathbf{u}^2 + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{u})^2 \right], \quad (2)$$

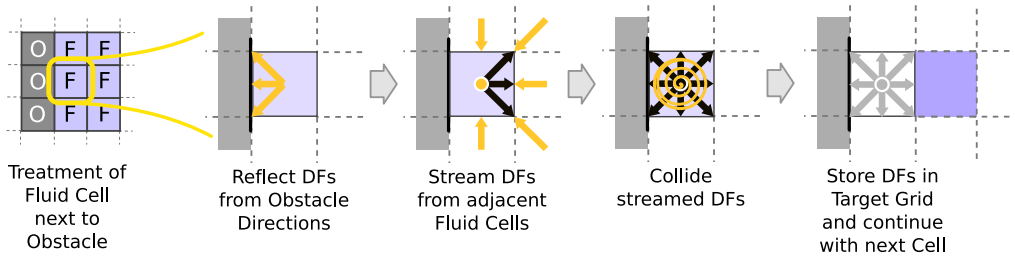


Figure 3. This figure gives an overview of the stream and collide steps for an exemplary cell next to an obstacle.

where the weights w_i are:

$$\begin{aligned} w_i &= 1/3 \quad \text{for } i = 1, \\ w_i &= 1/18 \quad \text{for } i = 2, \dots, 7, \\ w_i &= 1/36 \quad \text{for } i = 8, \dots, 19. \end{aligned}$$

The macroscopic fluid variables density and velocity are computed as the first two moments of the distribution functions for each cell

$$\rho = \sum_{\alpha=1}^{19} f_{\alpha} \quad \text{and} \quad \mathbf{u} = \sum_{\alpha=1}^{19} \mathbf{e}_{\alpha} f_{\alpha} . \quad (3)$$

Relaxing the DFs towards the equilibrium is performed with the relaxation time τ , which is calculated given the lattice viscosity ν as

$$\nu = \frac{\tau - 0.5}{3} . \quad (4)$$

The DFs for the next time step are then computed with the streamed DFs and the equilibrium distribution functions, calculated using velocity and density given by the streamed DFs, with

$$f_i(\mathbf{x}, t + \Delta t) = (1 - \omega) f'_i(\mathbf{x}, t + \Delta t) + \omega f_i^{eq} , \quad (5)$$

with $\omega = 1/\tau$.

This model is explained in more detail in e.g. [10]. It is commonly called *LBGK* model due to the simplification of the particle collisions with a linear relaxation time [1]. Note that density and velocity are not changed by the collision process. Hence, the streamed DFs, the equilibrium DFs and the collided DFs all have the same values for ρ and \mathbf{u} according to Eq. 3.

The standard boundary conditions for LBM are no-slip obstacles implemented with the bounce-back rule. During streaming all values that would move into a wall are

inverted and copied back to the originating cell. This is equivalent of changing Eq. 1 to:

$$f'_i(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x}, t), \quad (6)$$

and results in a zero tangential and normal velocity between fluid and obstacle cells. The algorithm so far can simulate the flow in domains completely filled with fluid and with arbitrary obstacles.

Given the real-world values for viscosity $\nu' [\frac{m^2}{s}]$, domain size $S [m]$, a desired grid resolution r and gravitational force $\mathbf{g}' [\frac{m}{s^2}]$ we compute the lattice values in the following way. For simplicity, we will assume that S is the length of one side of the domain, that should be resolved with r cells. Thus, the size of an LBM can be computed as

$$\Delta x = S/r. \quad (7)$$

The timestep Δt is computed by limiting the compressibility due to the gravitational force. In the following we have chosen a value of $g_c = 0.005$ to keep the compressibility error below half a percent. Thus,

$$\Delta t = \sqrt{\frac{g_c/r \cdot \Delta x}{|\mathbf{g}'|}} \quad (8)$$

yields a time step ensuring that the force exerted upon each cell due to the gravitational acceleration is causing less than a factor g_c of compression. Given Δx and Δt , the lattice viscosity ν is computed as

$$\nu = \nu' \frac{\Delta t}{\Delta x^2}, \quad (9)$$

and can be used to compute τ using Eq. 4. Likewise, the lattice acceleration \mathbf{g} is calculated as

$$\mathbf{g} = \mathbf{g}' \frac{\Delta t^2}{\Delta x}. \quad (10)$$

The following sections will describe extensions to the basic LBM described so far. We will introduce free surface handling, adaptive time step resizing, the Smagorinski turbulence model and a grid refinement algorithm. Section 3 will then explain how to couple these extensions to create an efficient and stable free surface fluid simulator.

2.1 Free Surfaces

To track free surfaces we introduce two additional cell types: *interface cells*, and *empty cells*. Empty cells are void of fluid, while partially filled interface cells are required to separate empty cells from fluid cells. Free surface boundary conditions are set for interface cells, which also store a fluid fraction value, similar to volume-of-fluid methods for conventional NS solvers [11]. Furthermore cell type conversions need to be handled if interface cells become completely filled or empty. The boundary conditions presented here do not compute the gas phase as a separate fluid, but assume a viscosity difference between gas and fluid phase that is high enough to approximate the gas velocity near the interface with the fluid velocity. This is especially suitable for simulations with large gas regions, as these do not require any computation. Empty cells that contain no fluid need not be considered in the algorithm until they are eventually converted to interface cells as described below. An outline of the free surface treatment is given in Fig. 4 .

The movement of the free surface is computed directly from the DFs, as these are the values that are actually advected during the streaming step. For each interface cell we additionally store the current mass m that it contains. The fluid fraction ε of the cell is computed with the mass value as

$$\varepsilon = m/\rho, \quad (11)$$

where ρ is computed with Eq. 3. For the mass exchange between two interface cells, their fluid fraction is taken into account to approximate the area they share at the cell boundary:

$$\Delta m_i(\mathbf{x}, t + \Delta t) = \left[f_i(\mathbf{x} + \mathbf{e}_i, t) - f_i(\mathbf{x}, t) \right] \frac{\varepsilon(\mathbf{x} + \mathbf{e}_i, t) + \varepsilon(\mathbf{x}, t)}{2}. \quad (12)$$

If the adjacent cell is a fluid cell, the mass exchange is simplified to

$$\Delta m_i(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x} + \mathbf{e}_i, t) - f_i(\mathbf{x}, t) \quad (13)$$

to match the DFs that are exchanged during the streaming step. For all interface cells, the value of m for the next time step is computed by summing the mass changes of all velocity directions before performing the streaming step:

$$m(\mathbf{x}, t + \Delta t) = m(\mathbf{x}, t) + \sum_{\alpha=1}^{19} \Delta m_{\alpha}(\mathbf{x}, t + \Delta t). \quad (14)$$

For fluid cells, the mass is equal to their density, the fluid fraction being $\varepsilon = 1$, while for empty or boundary cells no mass exchange needs to be considered.

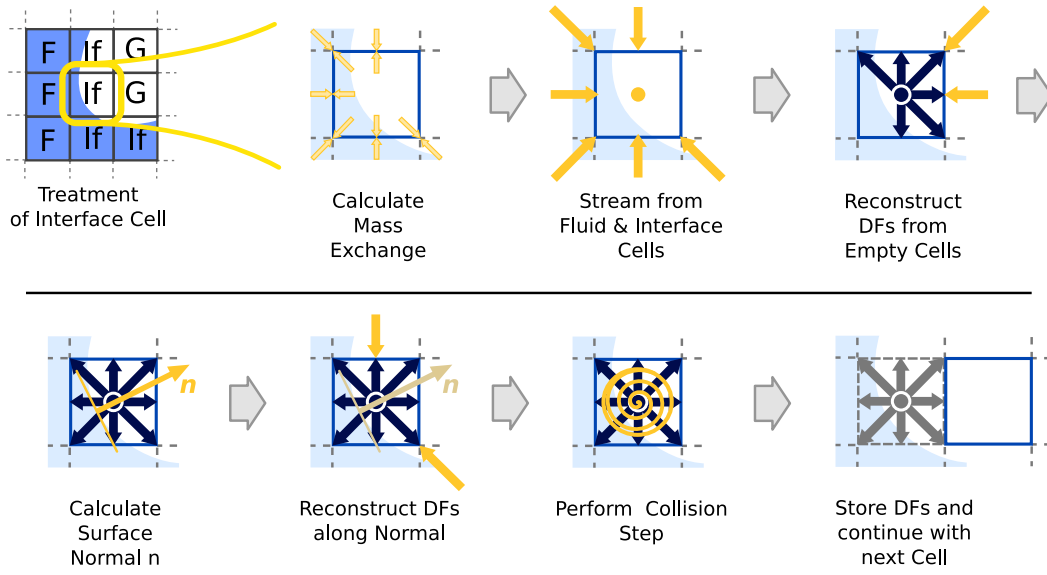


Figure 4. An illustration of the steps that have to be executed for an exemplary interface cell.

DFs in interface cells coming from the direction of an empty cell during streaming must be reconstructed to ensure correct interface movement and a valid set of DFs for interface cells. It is assumed, that the pressure in the gas phase and its density ρ_G is the same as reference pressure and density of the LBM simulation, hence $\rho_G = 1$. In terms of distribution functions, this means that for an interface cell at position \mathbf{x} with an empty cell at $(\mathbf{x} + \mathbf{e}_i)$ the streamed DF f'_i is reconstructed as:

$$f'_i(\mathbf{x}, t + \Delta t) = f_i^{eq}(\rho_G, \mathbf{u}) + f_i^{eq}(\rho_G, \mathbf{u}) - f_i(\mathbf{x}, t). \quad (15)$$

Here \mathbf{u} is the velocity of the interface cell. In this form the boundary conditions do not yet include effects such as surface tension or bubble pressure. These could, however, be included as a scaling factor of the two equilibrium distribution functions, for details see e.g. [25]. All DFs that would be streamed from empty cells are calculated with Eq. 15, in addition to the DFs coming from the half space given by the tangential plane of the fluid surface. The latter step is required to balance forces on both sides of the fluid gas interface. A surface normal is calculated by finite differences from the fluid fraction values to determine the velocity directions coming from the gas phase. The new set of DFs is now used to calculate the current cell density, and determine from the fluid fraction value whether the interface cell might have been filled ($\epsilon > 1$) or emptied ($\epsilon < 0$).

Once the stream step including interface cell treatment and the collide step have taken place, the cell type conversion of filled or emptied interface cells is carried out. When performing a cell type conversion from interface cell to empty or fluid cell, usually some excess mass needs to be redistributed to surrounding interface cells, as the interface cells often do not end up with exactly $m = \rho$ or $m = 0$ at the end of a time step. Furthermore, the layer of interface cells must remain closed. For

an emptied interface cell, all fluid cells in its neighborhood have to be converted to interface cells. Likewise empty cells must be converted to interface cells when interface cells in their neighborhood have become filled. Once all filled and emptied cells have been handled, the next LBM step is performed. Further details of the algorithm can be found in e.g. [26]. Overall, in addition to the high computational efficiency, the advantages of the algorithm are the completely local treatment of the free surface boundary conditions and the mass conservation up to machine precision.

2.2 Turbulence Model

In order to simulate high Reynolds number flows with the LBM, the basic algorithm needs to be extended as its stability is limited once the relaxation parameter τ approaches $1/2$. Various methods for stabilization have been developed. The two most popular approaches are the multi relaxation time method (MRT) and the Smagorinski sub-grid turbulence model. The former replaces the single time relaxation operator by a more advanced operator that relaxes the different hydrodynamic moments separately. The sub-grid model, as derived in [12], on the other hand models the effect of sub-grid scale vortices by modifying the viscosity according to the Reynolds stress tensor. The latter approach has the advantage of allowing very high Reynolds numbers, and has a wide applicability, while the parameters of the MRT models need to be tailored to specific flows in order to achieve the desired effect. In the following we will apply the Smagorinski sub-grid model, as used in e.g. [31]. The increase in stability allows the computation of turbulent flows with a relatively low grid resolution. Compared to the small slowdown due to the increased complexity of the collision operator this usually results in a large improvement of efficiency.

The sub-grid turbulence model applies the calculation of the local stress tensor as described in [22] to the LBM. This is simplified, as for LBM each cell already contains information about the derivatives of the hydrodynamic variables in each DF. The magnitude of the strain rate 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 Smagorinski constant C is used, for which we chose a value of 0.04. Values in this range are commonly used for LBM simulations, and were shown to yield good modeling of the sub-grid vortices [34]. The turbulence model is integrated into the basic algorithm as described in Section 2 by adding the calculation of the modified relaxation time after the streaming step, and using this value in the normal collision step that was described above.

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

for each cell with

$$\Pi_{i,j} = \sum_{\alpha=1}^{19} \mathbf{e}_{\alpha_i} \mathbf{e}_{\alpha_j} (f_i - f_i^{eq}), \quad (16)$$

where we have used the notation from [12]. Thus i and j each run over the three spatial dimensions, while α is the index of the respective velocity vector for the D3Q19 model.

As in [12], the intensity of the local strain tensor S is then computed as

$$S = \frac{1}{6C^2} \left(\sqrt{v^2 + 18C^2 \sqrt{\Pi_{i,j} \Pi_{i,j}}} - v \right). \quad (17)$$

Now the modified relaxation time is computed as

$$\tau_s = 3(v + C^2 S) + \frac{1}{2}. \quad (18)$$

From Eq. 17 it can be seen that 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 τ .

2.3 Adaptive Time Steps

Gravity driven flows such as the free surface flow of Fig. 1 are usually initialized by a fluid configuration and an gravitational force. The maximum velocities are often not a priori known, which makes it hard to parametrize LBM simulations and often leads to unnecessarily small time steps in combination with long computation times. The method described in this section dynamically changes the LBM parametrization according to the velocities [27]. As the size of the time step is not a parameter of the LBM equations it is only changed when necessary due to large or small velocities. This furthermore requires a recalculation of the LBM relaxation time and a rescaling of the DFs to match the new values for pressure and velocity according to the chosen time step size. In the following, a subscript of o will denote values before the time step change, while a subscript of n will indicate values for the new parametrization.

Given an initial simulation setup as described in Section 2 with a value for τ and an external force \mathbf{g} , the time step has to be reduced if the norm of the maximum velocity \mathbf{u}_{\max} exceeds a certain value:

$$|\mathbf{u}_{\max}| > \frac{1}{6}/\xi, \text{ with } \xi = \frac{4}{5}. \quad (19)$$

We use $1/6$ as the velocity threshold, as it is the half of $1/3$, at which point the DFs according to Eq. 2 would become negative. If Eq. 19 holds, the new time step size is given by

$$\Delta t_n = \xi \Delta t_o, \quad (20)$$

where Δt_o , the old step size is initially equal to 1. Once the fluid slows down, the time step could be increased again to $\Delta t_n = \Delta t_o / \xi$. As for LBM the value of τ also depends on the size of the time step, it changes according to:

$$\tau_n = s_t \left(\tau_o - \frac{1}{2} \right) + \frac{1}{2}, \quad \text{with } s_t = \Delta t_n / \Delta t_o. \quad (21)$$

The new acceleration for a LBM step is then calculated as

$$\mathbf{g}_n = s_t^2 \mathbf{g}_o. \quad (22)$$

To account for the new time step size, the velocity and also the density deviation from the median density ρ_{med} have to be rescaled for each cell. Hence, after calculating ρ_o and \mathbf{u}_o with Eq. 3 for an interface or fluid cell, the new values are computed with:

$$\begin{aligned} \rho_n &= s_t (\rho_o - \rho_{\text{med}}) + \rho_{\text{med}} \quad \text{and} \\ \mathbf{u}_n &= s_t \mathbf{u}_o, \end{aligned} \quad (23)$$

where the median density ρ_{med} is calculated from the total fluid volume V and the total mass M as $\rho_{\text{med}} = V/M$. The total volume is calculated by summing the values of ε over all cells, while M is the sum of all masses. The fill fraction and mass of interface cells are given by:

$$\begin{aligned} m_n &= m_o (\rho_o / \rho_n) \quad \text{and} \\ \varepsilon_n &= m_n / \rho_n, \end{aligned} \quad (24)$$

The non-equilibrium parts of the DFs determine the relaxation towards equilibrium state according to the relaxation time τ . When τ changes with the changing time step size, the fluid behavior should not be influenced by this reparametrization. Therefore the non-equilibrium parts have to be rescaled in a way that is similar to the rescaling procedure for grid refinement from [5]. Furthermore, the rescaled DFs have to match the new macroscopic quantities for velocity \mathbf{v}_n and pressure deviation ρ_n . DFs f_i^n for the new time step size are calculated with:

$$f_i^n = s_f \left[f_i^{eq}(\rho_o, \mathbf{u}_o) + s_\tau (f_i - f_i^{eq}(\rho_o, \mathbf{u}_o)) \right], \quad (25)$$

where s_f and s_τ are calculated as follows:

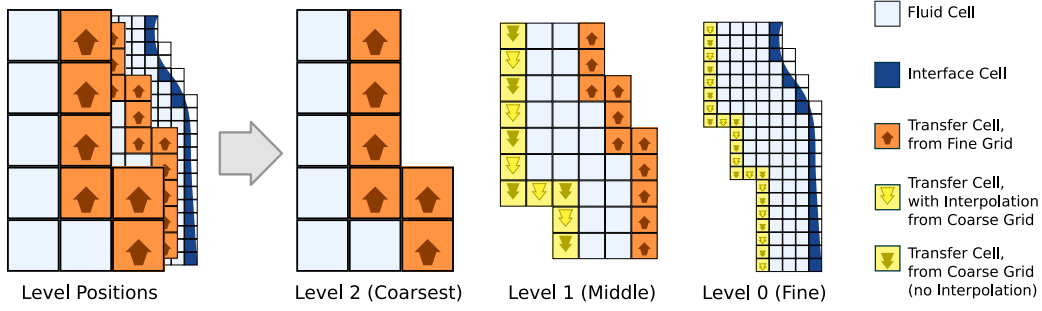


Figure 5. This picture shows an example of a coarsened fluid region near the free surface with 2 levels of coarser grids. To the left the transfer cell layers for coupling adjacent levels can be seen.

$$\begin{aligned} s_f &= f_i^{eq}(\rho_n, \mathbf{u}_n) / f_i^{eq}(\rho_o, \mathbf{u}_o) \\ s_\tau &= s_t(\tau_n / \tau_o). \end{aligned} \quad (26)$$

The rescaling procedure to change the time step size requires roughly the same computational effort as a normal collision step. As it is performed seldom in comparison to the number of LBM steps, it usually requires ca. 1% of the overall computation time, however, it can reduce the overall number of time steps significantly.

2.4 Grid Refinement

In [5], Filippova et. al. developed an algorithm to couple LBM 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. 25. 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 been used e.g. in [32] to compute simulations of an airfoil on a grid with refined blocks. Rohde recently proposed an alternative approach for grid refinement with LBM, see [21] for details. However, as this method requires an additional filtering step to ensure stability, our work is based on the algorithm described in [5].

Fig. 5 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 \mathbf{e}_i , with $f_{f,i}$ being its counterpart on the fine grid. As can be seen in Fig. 5, the grid spacing Δx_c and Δt_c on the coarse grid are twice those of the fine grid. According to Eq. 4 this means that the relaxation time needs to be calculated with the corresponding parameters for each grid. Reformulating Eq. 4

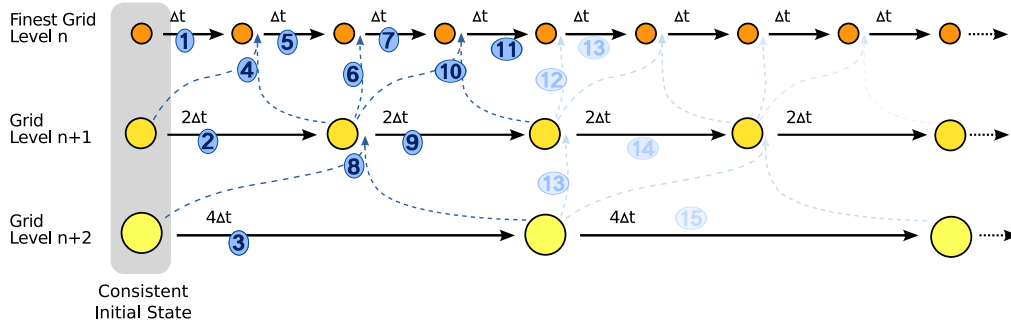


Figure 6. Here the effect of the different time step sizes for multiple simulation grids is shown. The numbers indicate the order in which the steps are performed. Dashed arrows indicate interpolation, while straight arrows from one circle to another represent LBM steps with the indicated time step length.

and 9 using $\Delta x_c = 2\Delta x_f$, the relaxation time for the coarse grid is calculated by

$$\tau_c = \frac{1}{2}(\tau_f - \frac{1}{2}) + \frac{1}{2}. \quad (27)$$

In Fig. 5 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

$$f_{c,i} = f_{f,i}^{eq} + s_{cf} [f_{f,i} - f_{f,i}^{eq}], \quad (28)$$

where s_{cf} is calculated as

$$s_{cf} = \frac{2(\tau_c - 1)}{(\tau_f - 1)}. \quad (29)$$

Note that Eq. 29 has a singularity for $\tau_f = 1$. However, this is unproblematic for turbulent flows, as the low viscosity will always result in values of τ close to $1/2$. For a transfer in the other direction, from the coarse to the fine grid, Eq. 28 becomes

$$f_{f,i} = f_{c,i}^{eq} + \frac{1}{s_{cf}} [f_{c,i} - f_{c,i}^{eq}]. \quad (30)$$

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. 28. 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. 28, the DFs of the coarse grid are first interpolated to compute the corresponding values at the position of the fine grid cell. As described in e.g. [32], 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 LBM steps, only a single one has to be performed on the coarse grid. This, however, means that for one of the two fine grid LBM 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 Fig. 6.

3 Adaptive Coarsening Algorithm

In this paper we take the view 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 following paragraphs. We will explain how to combine the free surface LBM method with the turbulence model and the adaptive time steps described above. Afterwards we will show 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.

3.1 Turbulence Model

The free surface extension of Section 2.1 and the Smagorinski model of Section 2.2 can be combined directly. The turbulence model differs from algorithms such as MRT since it does not change the equilibrium distribution function. Furthermore, the free surface equations in Section 2.1 are independent of the lattice viscosity. Thus, the boundary conditions and mass tracking formulas remain valid. The stability of the turbulence model is transferred directly to the free surface simulations, hence enabling the computation of free surface flows with high Reynolds numbers, and values of τ 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.

3.2 Adaptive Time Steps

The adaptive time step procedure from Section 2.3 can be used in order to avoid too large time steps causing instabilities. When the size of the time step is reduced to simulate large velocities, the value of τ becomes smaller according to Eq. 21.

Instabilities due to τ being almost 0.5 are alleviated by applying the turbulence model. This in turn requires a modification of Eq. 25, as the non-equilibrium scaling of the adaptive time steps depends on the relaxation time τ .

With Eq. 21, the lattice viscosities ν_n and ν_o for the old and the new time step are calculated. Eq. 16, 17 and 18 can then be used to compute the modified local relaxation times for each cell, $\tau_{s,n}$ and $\tau_{s,o}$, with ν_n and ν_o . Eq. 25 must be modified to include the local relaxation time from the turbulence model. This is done by calculating the scaling factor s_τ using the local relaxation times as

$$s_\tau = s_t(\tau_{s,n} / \tau_{s,o}). \quad (31)$$

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 LBM steps to compute the solution.

The following section 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 LBM step. This is an important component for a stable and highly efficient LBM free surface fluid simulator.






3.3 Adaptively Coarsened Grids

For dynamic problems, such as free surface flows or flows with moving obstacles, the techniques described in Section 2.4 cannot be applied without modifications. In [3] and [16] 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 such as those of Fig. 1, the region of interest, that needs to be accurately computed is the free surface itself. Hence, we perform the simulation of this surface on a fine computational grid, while the accuracy of the computation inside of the fluid may be less important. In the following we will describe 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. 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 e.g. smooth free surface regions with few details. However, this would cause problems for the mass conservation with the mass flux given by Eq. 12 and make generating a triangulated surface more complicated.

We thus ensure 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 [29], we will denote the fine to coarse grid transfer with *restriction* and the coarse to fine grid transfer with *prolongation* in the following sections.

3.3.1 Boundary Cell Conversion

To adapt the coarse grids to the movement of the free surface, while keeping the transfer cell layers consistent, we have developed a set of rules to determine when to refine or coarsen a grid region. 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 its neighborhood and its neighborhood 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 5. Hence, we have decided to explain and implement each pass as a separate sweep over the cell flags. In the following, we will distinguish five cell types:

-  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.

The following rules are applied to all coarse levels. For the first level of coarsening, we ensure 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. In the following explanation we can 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 2.4. 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.

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 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 following 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 directly turned into unused cells. Fine grid cells

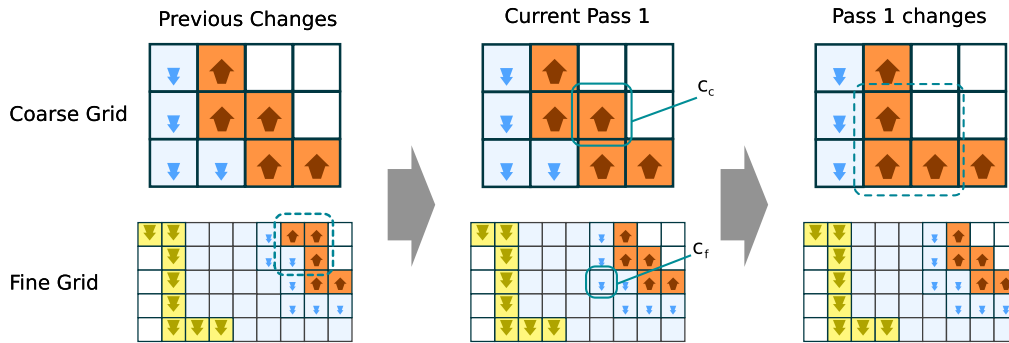


Figure 7. Pass 1.

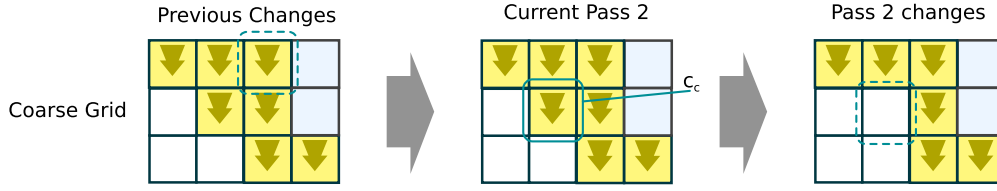


Figure 8. Pass 2.

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 we apply linear instead of second-order spatial interpolation 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 4 we will provide evidence that the accuracy of the linear interpolation is computationally sufficient by comparing it directly to a second order interpolation.

3.3.2 Grid Transfer

These conversion rules are checked before each coarse grid LBM 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 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 LBM step.

The transfer of DFs on the boundaries is done by including the modified relaxation time of the turbulence model in Eq. 28. After interpolation of the DFs, the modified

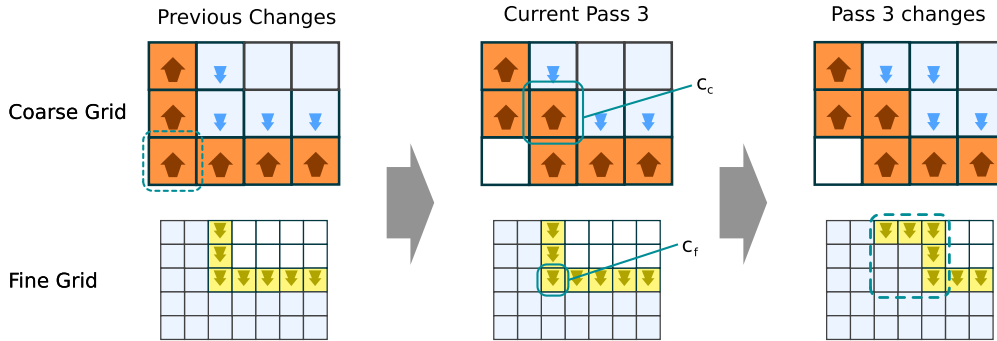


Figure 9. Pass 3.

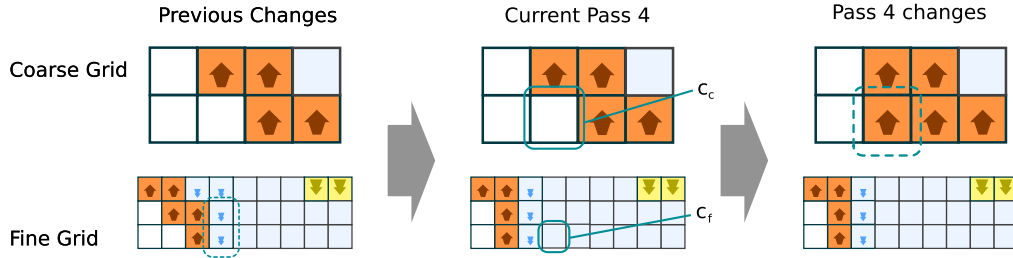


Figure 10. Pass 4.

relaxation times τ_{sc} and τ_{sf} are calculated with Eq. 29 using the viscosities ν_c and ν_f , respectively. Finally, the scaling factor s_{cf} is calculated with

$$s_{cf} = \frac{2(\tau_{sc} - 1)}{(\tau_{sf} - 1)}, \quad (32)$$

and used instead of Eq. 29 with Eq. 28 and Eq. 30.

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

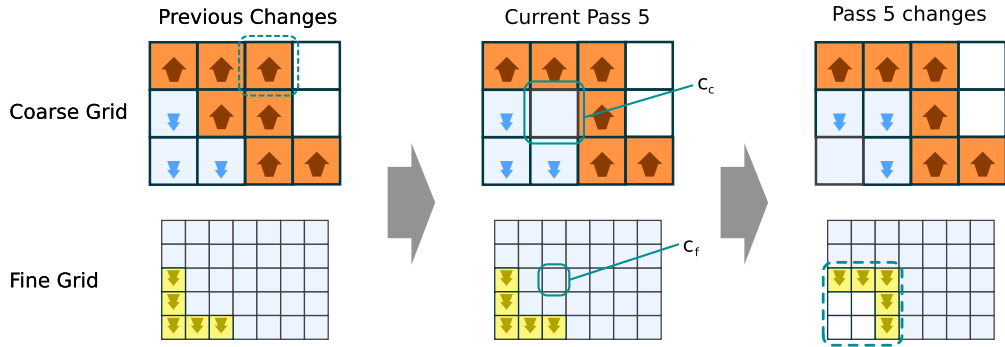


Figure 11. Pass 5.

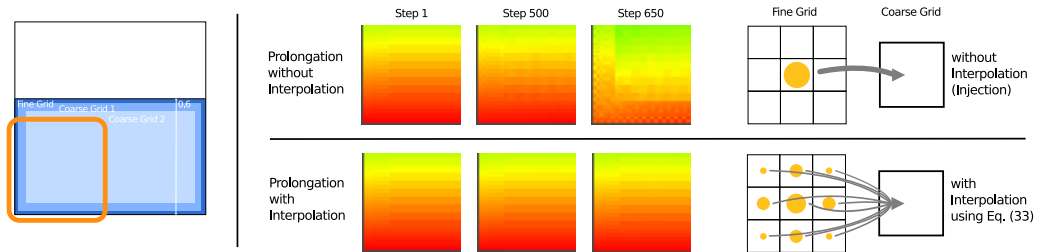


Figure 12. Example of artifacts that occur for a simple standing fluid test case with a resolution of 128^2 and two coarse levels. Each picture shows the density distribution in the lower left corner of the fluid, where green values indicate $\rho = 1.0$ while a red color indicates larger values. The upper row of pictures was created without any interpolation for the prolongation, while the lower row makes use of Eq. 33.

example of this problem can be seen in Fig. 12. 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 2.4 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, we use a restriction that takes into account all fine grid cells within the fine grid neighborhood of a coarse grid cell, as shown on the right side of Fig. 12 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. 30 are calculated as

$$\tilde{f}_{f,i}(\mathbf{x}) = \sum_{\alpha=1}^{19} f_{f,i}(\mathbf{x} + \mathbf{e}_{\alpha}) \frac{w_{\alpha}}{w_{\text{total}}} \quad (33)$$

with

$$w_{\alpha} = e^{-|\mathbf{e}_{\alpha}|} - e^{-2.3}, \quad w_{\text{total}} = \sum_{\alpha=1}^{19} w_{\alpha} \quad (34)$$

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

In conclusion, our 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 LBM step (for the finest level this includes handling the free surface).
- (5) Continue with the next finer grid.

We will evaluate the accuracy of both the interpolation scheme and the adaptive coarsening algorithm in the following section.

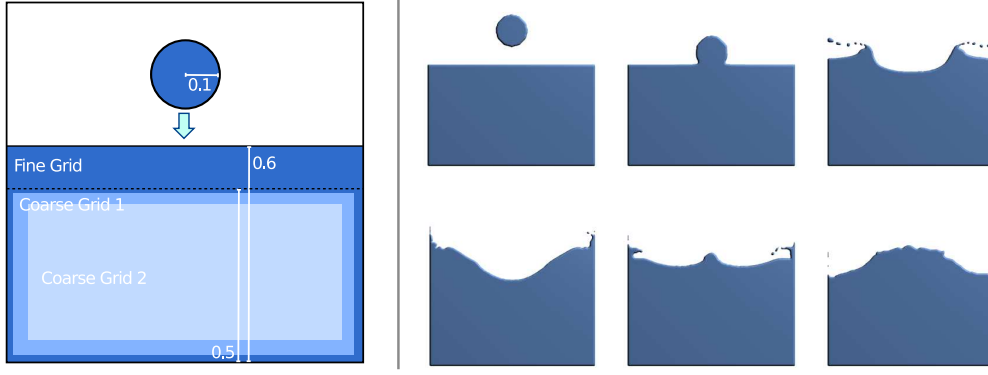


Figure 13. Falling drop test case setup for interpolation accuracy measurements with static coarsening.

4 Validation

The accuracy of the different grid transfer methods will be determined by comparing \mathcal{E} , which is the average deviation of the fluid fraction values ε 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, its value will be close to one, while values close to zero indicate a similar shape of the fluid. We normalize the measurements 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 values shown in Fig. 14 and Fig. 16 are thus computed as

$$\mathcal{E} = \frac{1}{t_{\text{total}}} \frac{1}{n_{\text{total}}} \sum_{t=1}^{t_{\text{total}}} \sum_{\mathbf{x} \in \Omega} |\varepsilon_{\text{ref}}(\mathbf{x}, t) - \varepsilon(\mathbf{x}, t)|, \quad (35)$$

where ε_{ref} are the fluid fraction values of the corresponding fine-resolution reference simulation, Ω is the size of the domain ranging from 0 to 1 in each spatial dimension, and t_{total} is the number of timesteps to average over. Likewise, n_{total} is the total number of chosen points where \mathcal{E} is measured at. For Fig. 14 the grid resolution of the reference simulation was used to set the number of measurement points. Note that \mathcal{E} in contrast to e.g. error metrics from the multi grid literature does not 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, we chose $\nu' = 1^{-6} [\text{m}^2/\text{s}]$ and an acceleration of $\mathbf{g}' = (0, -9.81, 0)^T [\text{m}/\text{s}^2]$.

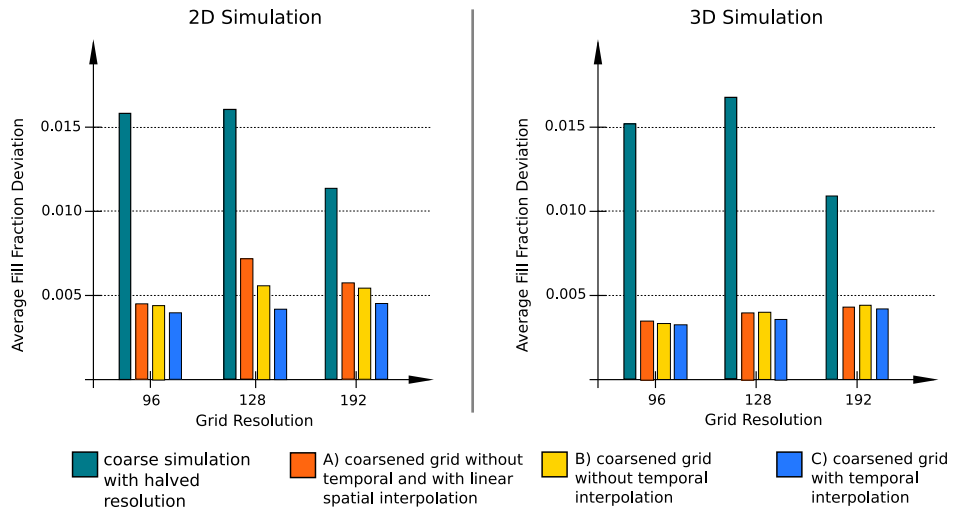


Figure 14. Accuracy measurement for the interpolation test case of Fig. 13.

4.1 Static test case

The different interpolation methods will be tested using the simulation setup described on the left side of Fig. 13. In Fig. 13 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 192^2 can be seen on the right side of Fig. 13. In three dimensions, the drop is converted to a cylinder with rounded edges, to match the two dimensional test case. As indicated in Fig. 13 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.

Fig. 14 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 25% 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).

Throughout the runs it can be seen, that the adaptively coarsened simulations are

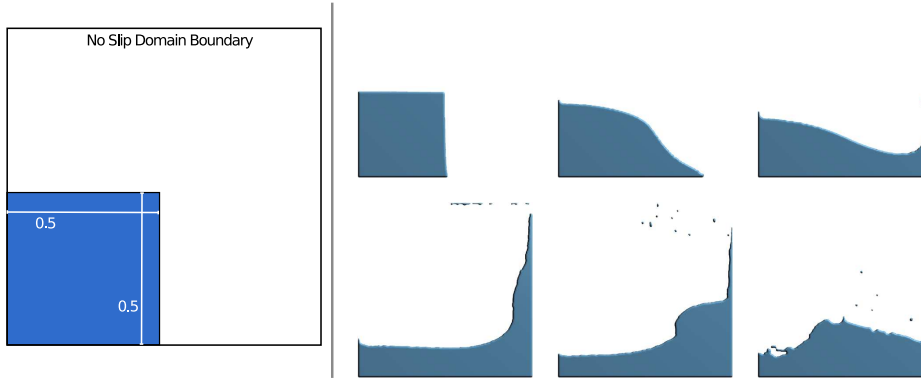


Figure 15. Breaking dam test case setup for accuracy measurements for the adaptive coarsening procedure.

significantly more accurate than the one run 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 [33], 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 [19] 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.

4.2 *Dynamic test case*

To validate the accuracy of the adaptive coarsening technique described in Section 3 we have used the breaking dam setup described in Fig. 15. 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. 35 are shown in Fig. 16. 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 those of the previous test case. However, throughout the runs they are more accurate than the coarse simulation, while requiring significantly less LBM cells than the fine simulation and yielding the same amount of surface details. The following section will show performance results of simulation runs, to illustrate the speedup that is possible using our adaptive coarsening method.

5 Performance

Before analyzing the overall performance, it is important to know how the workload is distributed between the different parts of the algorithm. We have therefore profiled a run of the test case shown in Fig. 17 with a resolution of 256^3 on a single Opteron CPU for 2500 LBM steps. The adaptive coarsening algorithm was used for 3 coarse levels in addition to the finest one.

As can be seen in Table 1, the majority of the computations are necessary for advancing the finest grid and computing the free surface boundary conditions. The adaptive coarsening itself requires more computational effort than the LBM 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 LBM cell update.

Usually, the performance of LBM programs is measured with the number of cell

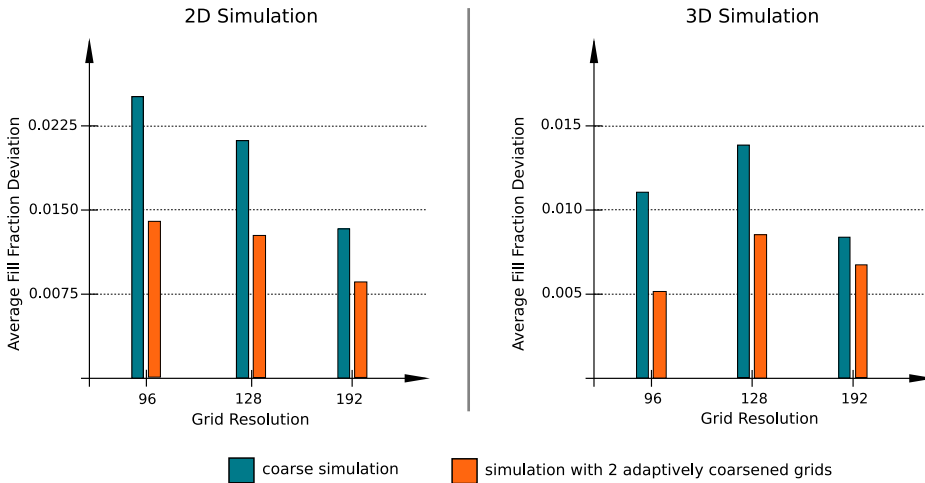


Figure 16. Accuracy measurement for the adaptive coarsening test case of Fig. 15.

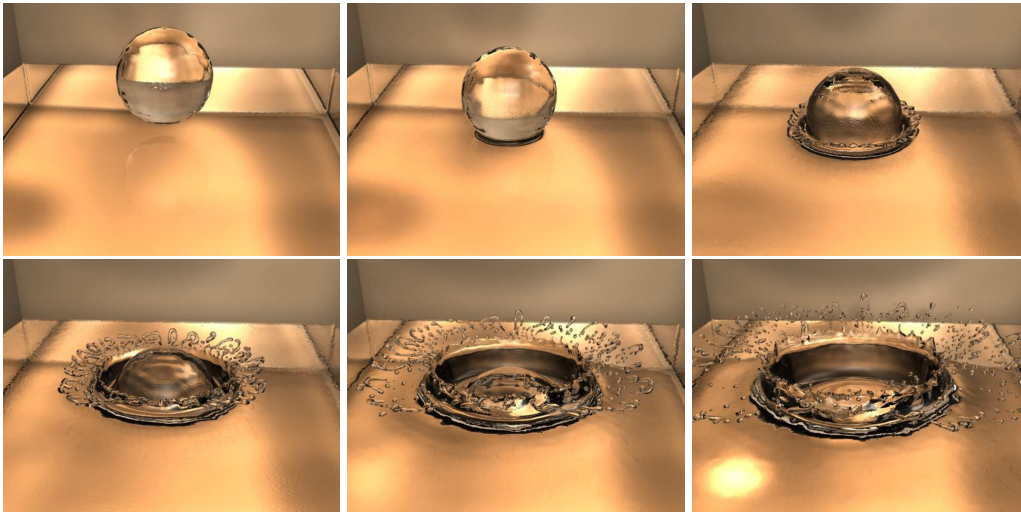


Figure 17. Images of the falling drop simulation with a grid resolution of 480^3 and the adaptive coarsening algorithm.

Table 1

Workload distribution for an exemplary simulation.

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

updates per second: *MLSUPS* (million lattice site updates per second). However, this is of course not valid anymore once coarsened grids are involved. 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 our implementation without adaptive coarsening for a falling drop test case as shown in Fig. 17.

Table 2 shows that our basic implementation yields a high performance on a variety of CPU architectures. This is important, as a poor implementation of the basic algorithm might yield larger speedups when combined with our adaptive coarsening technique – even when the overall performance is bad. In the following we will demonstrate the achievable speedups with the test cases shown in

- (1) Fig. 17, a falling drop test case, and
- (2) Fig. 21, 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

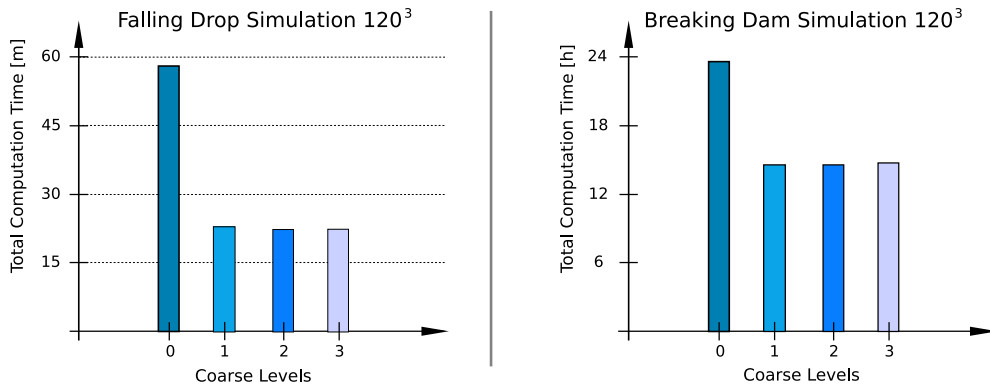


Figure 18. Performance for a resolution of 120^3 on a single Pentium4 CPU with 3.2GHz.

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 Fig. 18 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 our adaptive coarsening technique.

Fig. 19 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. In contrast to the 120^3 runs, more than a single level of coarsening yields a further speedup for test case A. In total, a speedup of 4.14 and 2.75 is achieved for test case A and B, respectively.

Table 2

Performance measurements of the basic free surface simulation code without adaptive coarsening on different architectures with up to four processors.

CPU	MLSUPS
Opteron 2.2 GHz (OpenMP version)	1.24
Opteron 2.2 GHz	1.55
Pentium4 3.2 GHz	1.84
Athlon64 2.4 GHz	1.98
Dual-Opteron (OpenMP version)	2.08
Quad-Opteron (OpenMP version)	3.73

The last performance results of Fig. 20 are for a resolution of 480^3 on a quad 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.

To allow the setup of more complicated simulation problems, and demonstrate the ability of our implementation to efficiently simulate free surface animations, we have implemented an interface to the 3D software *Blender*. Since version 2.40 the free surface simulation is part of the package, and can be obtained from [30]. It allows the productions of high quality fluid animations as shown in Fig. 22. The sources for the solver including the implementation of our adaptive coarsening algorithm were released under the *GNU Public License*, and is available on the the same website.

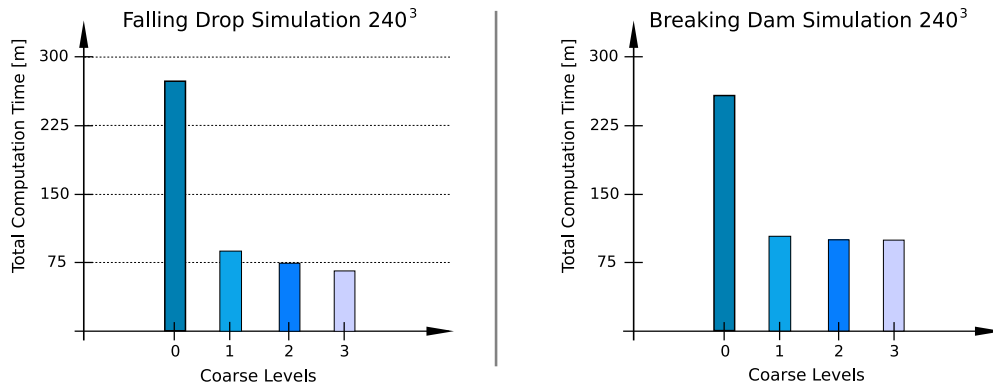


Figure 19. Performance with OpenMP parallelization for a resolution of 240^3 on a dual Opteron Node (each CPU with 2.2GHz).

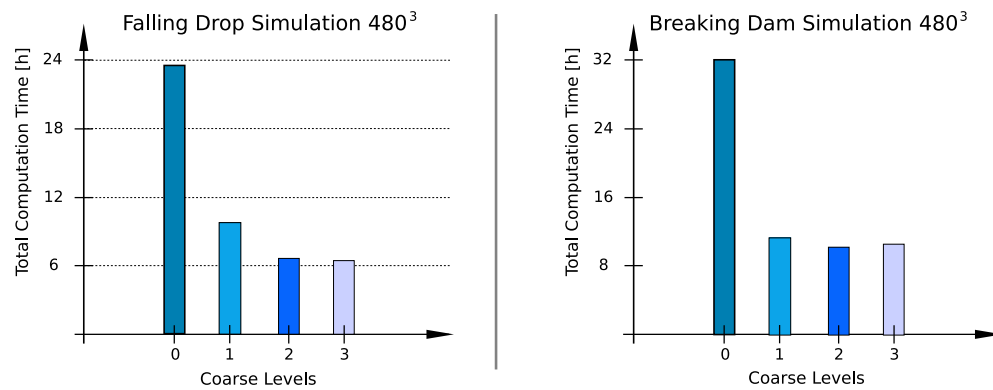


Figure 20. Performance with OpenMP parallelization for a resolution of 480^3 on a quad Opteron Node (each CPU with 2.2GHz).

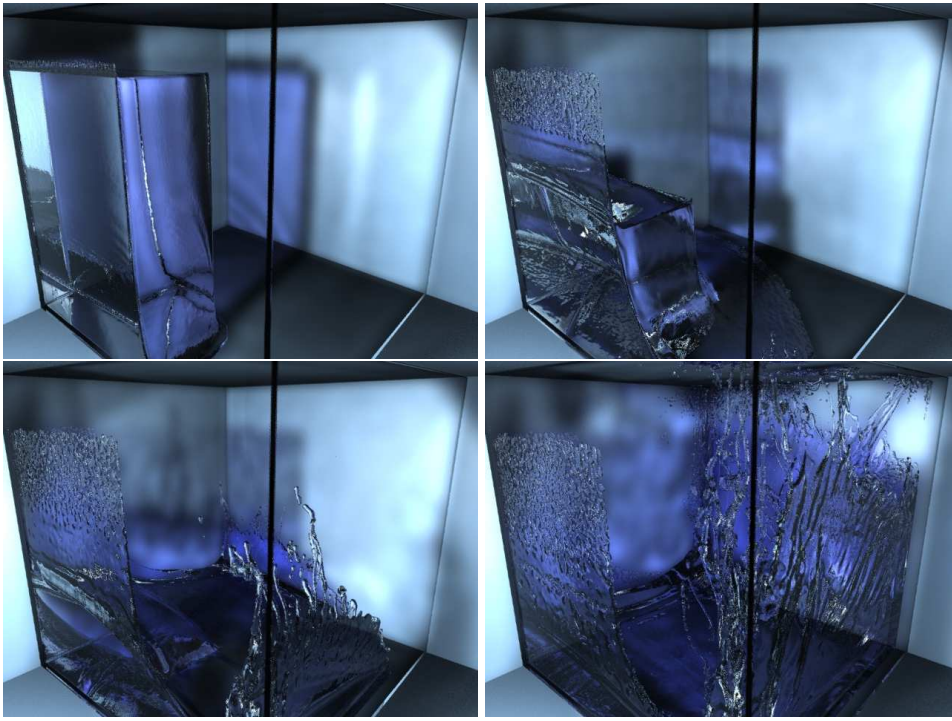


Figure 21. Pictures of the breaking dam test case, again with a grid resolution of 480^3 .

6 Conclusions & Outlook

We have presented a stable, accurate method for free surface simulations with the LBM, that can be used to efficiently simulate systems with large volumes of fluid. This is achieved by an algorithm to adaptively coarsen the simulation resolution inside of the fluid volumes. A set of rules is used to dynamically adapt the coarsened regions to the movement of the free surface. The combination with a Smagorinski turbulence model and an adaptive time step algorithm ensures stability of the fluid simulator. We have validated the algorithm comparing it to a fine grid simulation for static and dynamic test cases. The performance was evaluated with two different simulation setups and various grid sizes. Depending on the architecture and amount of fluid in the simulation, speed up factors of more than 3.5 are possible in comparison with a simulation on a single fine grid.

One area of future work will be to not only reduce the computational time but also to reduce the amount of memory. In our current implementation we allocate all simulation grids throughout the computational domain. Hence, we are planning to adaptively allocate patches in the fluid region for each grid level separately. This should significantly decrease the required memory, as coarsened regions inside of the fluid only have to store the coarsest grid level. It might, however, decrease the performance due to increased overhead of the patch management. Another challenge will be to efficiently parallelize our algorithm to run on large distributed memory systems. The basic LBM with a free surface extension already requires strategies for load balancing due to the movement of the fluid throughout the do-

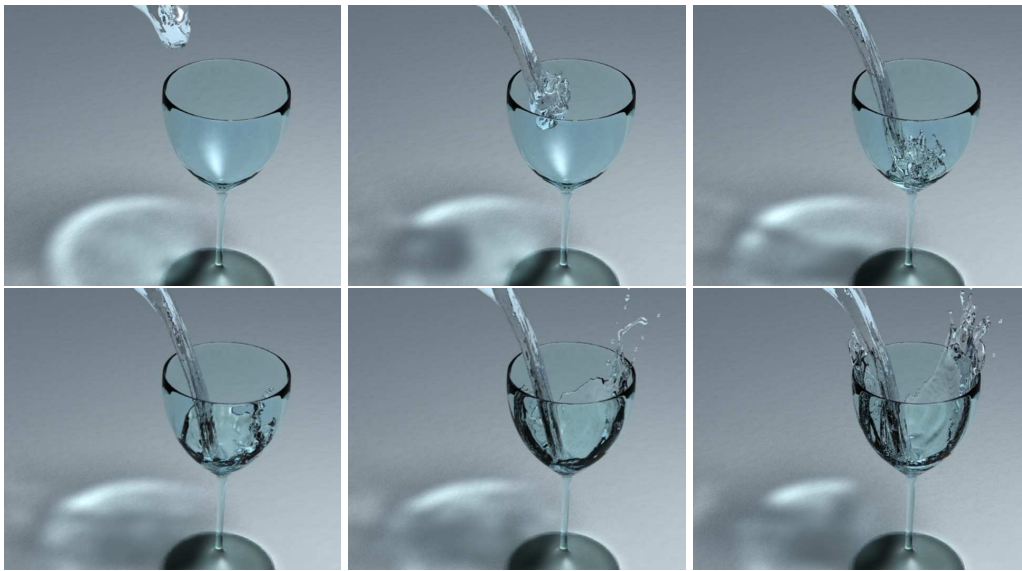


Figure 22. Several frames of animation from a simulation of filling a glass shaped obstacle.

main. With our adaptive coarsening algorithm the load balancing will also have to deal with balancing the load over all existing grids. Furthermore, our algorithm is applicable to simulations with moving obstacles by treating their boundaries as regions for refinement like the free surface.

As the motion of the free surface is naturally hard to estimate from still pictures, we have made the animations corresponding to Fig. 17,21 and 22 available on our website [28].

7 Acknowledgements

This research is funded by the DFG Graduate College GRK-244 *3-D Image Analysis and Synthesis*.

References

- [1] P. L. Bhatnagar, E. P. Gross, and M. Krook. A model for collision processes in gases. *Phys. Rev.*, 94, 1954.
- [2] Vivek Buwa, Daniel Gerlach, and Franz Durst. Regimes of bubble formation on submerged orifices. *Phys. Rev. Letters*, 2005.
- [3] B. Crouse, M. Krafczyk, J. Toelke, and E. Rank. A LB-based approach for adaptive flow simulations. *Int. J. of Modern Phys. B*, 17, 2003.
- [4] Ronald P. Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows. *J. of Comp. Phys.*, 152:457 – 492, 1999.

- [5] O. Filippova and D. Hänel. Grid Refinement for Lattice-BGK models. *J. of Comp. Phys.*, 147, 1998.
- [6] 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.
- [7] 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 & Fluids*, 2004.
- [8] D. Gerlach, G. Tomar, G. Biswas, and F. Durst. Comparison of volume-of-fluid methods for surface tension-dominant two-phase flows. *Int. J. of Head and Mass Transfer*, 2005.
- [9] X. He and L.-S. Luo. A Priori Derivation of Lattice Boltzmann Equation. *Phys. Rev. E*, 55, 1997.
- [10] X. He and L.-S. Luo. Lattice Boltzmann model for the incompressible Navier-Stokes equations. *J. of Stat. Phys.*, 88, 1997.
- [11] C. W. Hirt and B. D. Nichols. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *J. of Comp. Phys.*, 39, 1981.
- [12] 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.
- [13] 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.
- [14] C. Körner and R.F. Singer. Numerical Simulation of Foam Formation and Evolution with Modified Cellular Automata. *Metal Foams and Porous Metal Structures*, pages 91–6, 1999.
- [15] C. Körner, M. Thies, and R. F. Singer. Modeling of Metal Foaming with Lattice Boltzmann Automata. *Advanced Engineering Materials*, 2002.
- [16] Manfred Krafczyk. Gitter-Boltzmann-Methoden - von der Theorie zur Anwendung. Habilitation, 2001.
- [17] M. Krafczyk and J. Tölke and E. Rank and M. Schulz. Two-Dimensional Simulation of Fluid-Structure Interaction using Lattice-Boltzmann Methods. *Computers and Structures*, 79, 2001.
- [18] J. Monaghan. Smoothed particle hydrodynamics. *Annu. Rev. Astron. Phys.*, 30, 1992.
- [19] 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.
- [20] 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 *Proceedings of Supercomputing Conference 2004*, 2004.
- [21] M. Rohde. Extending the lattice-Boltzmann method: Novel techniques for local grid refinement and boundary conditions, PhD thesis. Technical report,

- 2004.
- [22] J. Smagorinski. General circulation experiments with the primitive equations. *Mon. Wea. Rev.*, 91, 1963.
 - [23] Jos Stam. Stable Fluids. *Proceedings of SIGGRAPH*, 1999.
 - [24] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001. 2 copies.
 - [25] N. Thürey. A Lattice Boltzmann method for single-phase free surface flows in 3D. Masters thesis, 2003.
 - [26] N. Thürey, C. Koerner, and U. Rüde. Interactive Free Surface Fluids with the Lattice Boltzmann Method, Technical Report 05-4. Technical report, Department of Computer Science 10 System Simulation, 2005.
 - [27] N. Thürey, T. Pohl, U. Rüde, M. Oechsner, and C. Koerner. Optimization and Stabilization of LBM Free Surface Flow Simulations using Adaptive Parameterization. *to appear in Computers and Fluids*, 2005.
 - [28] N. Thürey and U. Rüde. Webpage: Turbulent Free Surface Flows with the Lattice Boltzmann Method on adaptively coarsened Grids, 2005. <http://www10.informatik.uni-erlangen.de/~sinithue/turbfsflows/>.
 - [29] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.
 - [30] B. Veldhuizen, J. Langlotz, et al. Blender open source 3D graphics creation, 2005. <http://www.blender3d.org>.
 - [31] Xiaoming Wei, Ye Zhao, Zhe Fan, Wei Li, Suzanne Yoakum-Stover, and Arie Kaufman. Natural phenomena: Blowing in the wind. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation SCA '03*, July 2003.
 - [32] 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.
 - [33] 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.
 - [34] H. Yu, S.S. Girimaji, and Li-Shi Luo. Lattice Boltzmann simulations of decaying homogeneous isotropic turbulence. *Phys. Rev. E*, 71, 2005.