Comput Visual Sci (2009) 12:247–263
DOI 10.1007/s00791-008-0090-4

REGULAR ARTICLE

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

Nils Thürey · Ulrich Rüde

Received: 30 May 2006 / Accepted: 19 November 2007 / Published online: 18 March 2008

Abstract In this paper we will present an algorithm to perform free surface flow simulations with the lattice Boltzmann method on adaptive grids. This reduces the required computational time by more than a factor of three for simulations with large volumes of fluid. To achieve this, the simulation of large fluid regions is performed with coarser grid resolutions. 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. Furthermore, the free surface treatment is combined with a Smagorinsky turbulence model and a technique for adaptive time steps to ensure stable simulations. The method is validated by comparing the position of the free surface with an uncoarsened simulation. It yields speedup factors of up to 3.85 for a simulation with a resolution of 480^3 cells and three coarser grid levels, and thus enables efficient and stable simulations of free surface flows, e.g. for highly detailed physically based animations of fluids.

Keywords Free surface flows · Physically based animation · Adaptive grids · Lattice Boltzmann method

Communicated by G. Wittum.

N. Thürey · U. Rüde (✉)
Computer Science 10 - System Simulation (LSS),
University of Erlangen-Nuremberg, Cauerstr. 6,
91058 Erlangen, Germany
e-mail: Ulrich.Ruede@cs.fau.de

N. Thürey
e-mail: Nils.Thuerey@cs.fau.de

1 Introduction

Free surface flows are important for a variety of applications, such as the optimization of production processes for foaming or casting [23], the research of bubble formation regimes [4] or for applications in civil engineering such as certain types of fluid structure interactions [27]. It is furthermore of importance for physically based animations in computer graphics, as a realistic fluid motion is hard to achieve without relying on the equations that govern its motion. However, for all simulation problems that appear in these cases it is still problematic to ensure stability and reasonable computation times for complex flows.

Our fluid simulation uses the *lattice Boltzmann method* (LBM) which can efficiently handle irregular fluid geometries and topologies [11,32]. In contrast to solvers that directly compute solutions for the discretized Navier-Stokes equations, the LBM is a form of cellular automaton. It relaxes the incompressibility criterion and thus does not require an additional step to compute the pressure with an iterative method such as a multi-grid solver [5,53], or a pressure projection step [26,37]. Free surface fluids can also be computed with an approach known as smoothed particle hydrodynamics [29,30], which does not require a fixed grid and computes the fluid properties by computation kernels defined on particle neighborhoods. The LBM, however, is interesting due to the simple nature of the basic algorithm and its high efficiency. These properties of the algorithm make it possible to e.g. perform interactive fluid simulations [51], or adapt it to other problems [6,10,50]. The algorithm that we will present in this paper is based on the free surface algorithm that was developed to simulate metal foams [41]. The approach is similar to volume-of-fluid methods, that are often used in cases where mass conservation has to be guaranteed [17,39]. It furthermore does not require a simulation of the gas phase,

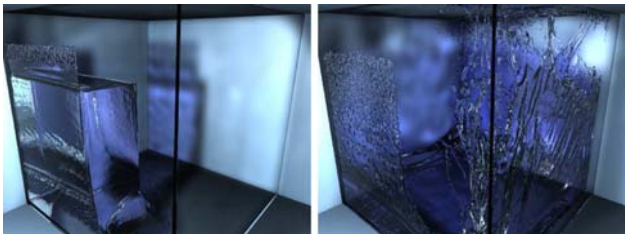

Fig. 1 An example of a free surface simulation created with the method described in this paper

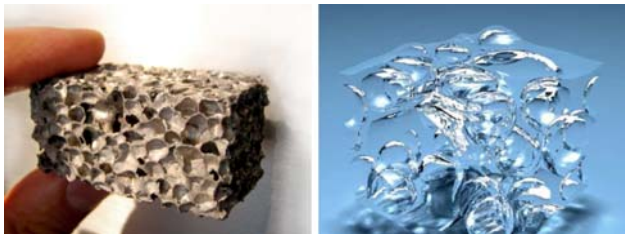

Fig. 2 Apart from physically based animations, the simulation of metal foaming processes is another possible application of the methods presented in this paper. To the *left*, an actual metal foam sample can be seen, while the *right* picture shows a foaming simulation performed with LBM in cooperation with C. Körner (*WTM* Erlangen)

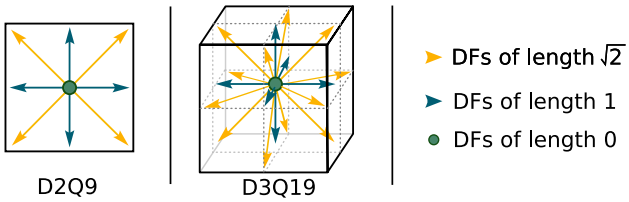

Fig. 3 The most commonly used LBM models in two and three dimensions

and thus saves significant amounts of work for cases with large gas regions [20].

We will first give an overview of the basic algorithm and its extensions. This will include the free surface boundary treatment, a subgrid 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. As we consider the algorithm to optimize a given simulation with a fine grid resolution, we will in the following refer to it as a *coarsening* algorithm, although it also requires the refinement of regions, to account for the movement of the free surface. In Sect. 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. Moreover, visualizations of these simulations with raytracing will be shown.

2 The lattice Boltzmann method

The LBM 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–Enskog expansion from statistical physics [9], or by direct discretization of the Boltzmann equation [15]. A more detailed overview of the basic algorithm together with extensions and applications can be found e.g. in [38] or in [54].

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 (Fig. 1). The velocity vectors $\mathbf{e}_1, \ldots, \mathbf{e}_9$ of the D2Q9 model, and $\mathbf{e}_1, \ldots, \mathbf{e}_{19}$ of the D3Q19 model are shown in Fig. 3. For each velocity vector a particle distribution function (DF) is stored. A DF f_i represents an amount of fluid moving with the velocity $\mathbf{e}_i$. The velocities of the D3Q19 model are

$$\begin{aligned} \mathbf{e}_1 &= (0,0,0)^T, \\ \mathbf{e}_{2,\ldots,7} &= (\pm 1,0,0)^T, (0,\pm 1,0)^T, (0,0,\pm 1)^T, \\ \mathbf{e}_{8,\ldots,11} &= (\pm 1,\pm 1,0)^T, \\ \mathbf{e}_{12,\ldots,15} &= (0,\pm 1,\pm 1)^T \quad \text{and} \\ \mathbf{e}_{16,\ldots,19} &= (\pm 1,0,\pm 1)^T. \end{aligned} \tag{1}$$

Thus there are particles not moving at all (f_1), moving with speed 1 ($f_2, \ldots, f_7$) and moving with speed $\sqrt{2}$ ($f_8, \ldots, f_{19}$). 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 1 in lattice units. The normalization procedure is explained below in more detail.

The basic LBM consists of two steps, the stream- and the collide-step. An overview of the two steps of the algorithm is given in Fig. 4. Here the streaming step represents the advection of the particles in the fluid. Post-streaming DFs f_i' thus can be written as:

$$f_i'(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x} - \Delta t \mathbf{e}_i, t). \tag{2}$$

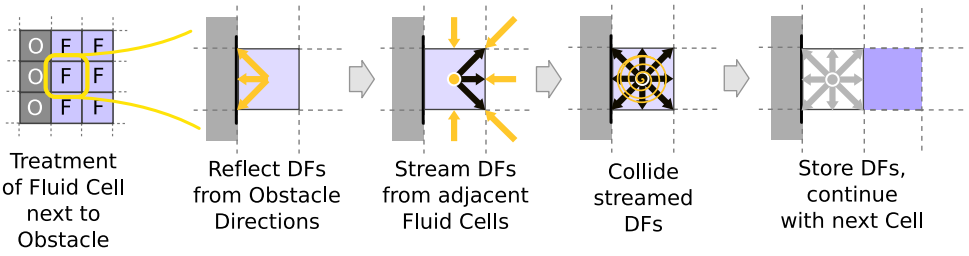

Fig. 4 This figure gives an overview of the stream and collide steps for a cell next to an obstacle

As Δx and Δt are both equal to one 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 post-streaming DFs of a cell with density ρ and fluid velocity $\mathbf{u}$ towards the equilibrium distribution function:

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

where the weights w_i are: $w_i = 1/3$ for $i = 1$, $w_i = 1/18$ for $i = 2, \ldots, 7$, and $w_i = 1/36$ for $i = 8, \ldots, 19$. The macroscopic fluid variables density and velocity are computed as the first two moments of the distribution functions for each cell

$$\rho = \sum_{i=1}^{19} f_i \quad \text{and} \quad \mathbf{u} = \sum_{i=1}^{19} \mathbf{e}_i f_i. \tag{4}$$

Here we use the incompressible model as described in [16], which alleviates compressibility effects of the standard model [33], by using a modified equilibrium distribution function and velocity calculation. Relaxing the DFs towards the equilibrium is performed with the relaxation time τ that takes values in the range from zero to two. It is given by the kinematic viscosity, see Eq. (8) below. The DFs for the next time step are then computed with the post-streaming DFs and the equilibrium distribution functions, calculated using velocity and density given by the post-streaming DFs, with

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

with $\omega = 1/\tau$.

This model is explained in more detail in e.g. [16]. It is commonly called *LBGK* model due to the simplification of the particle collisions with a single relaxation time [2,33]. Note that density and velocity are not changed by the collision process. Hence, the post-streaming DFs, the equilibrium DFs and the post-collision DFs all give the same values for ρ and $\mathbf{u}$ according to Eq. (4).

The simplest 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. (2) to:

$$f_i'(\mathbf{x}, t + \Delta t) = f_{\bar{i}}(\mathbf{x}, t), \tag{6}$$

and results in a zero tangential and normal velocity between fluid and obstacle cells. It has been shown, e.g., in [12] that the actual position of the boundary depends on the chosen lattice viscosity. This can be overcome by using the *multi relaxation time* method (MRT, [25]) that, in contrast to the single relaxation time described above, relaxes the different hydrodynamic moments individually. Moreover, various models for higher order no-slip boundary conditions are available, e.g., [3,28,49], and [12], as the bounce-back scheme only yields first order accuracy for arbitrary obstacles.

In the following we will describe the conversion of dimensional quantities, denoted by primed symbols, into dimensionless quantities used in the LBM. Given the real-world values for viscosity ν' (m^2/s), domain size S (m), a desired grid resolution r and gravitational force $\mathbf{g}'$ (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 cell size used by the LBM can be computed as $\Delta x' = S/r$. The dimensional time step $\Delta t'$ 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,

$$\Delta t' = \sqrt{\frac{g_c \cdot \Delta x'}{|\mathbf{g}'|}} \tag{7}$$

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 $\Delta x'$ and $\Delta t'$, the lattice viscosity ν and relaxation time τ are computed as

$$\nu = \nu' \frac{\Delta t'}{\Delta x'^2}, \quad \text{and} \quad \tau = 3\nu + 1/2. \tag{8}$$

Likewise, the lattice acceleration $\mathbf{g}$ is calculated as

$$\mathbf{g} = \mathbf{g}' \frac{\Delta t'^2}{\Delta x'}. \tag{9}$$

The following sections will describe extensions to the basic LBM described so far. We will introduce free surface handling, adaptive time step resizing, the subgrid 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 [17]. 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, since these do not require any computations. 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. 5 . While the VOF free surface model applied here was developed for the simulation of metal foams [21], without the need to explicitly simulate the gas phase, other multi-phase LBM approaches have been developed. In [14], Gunstensen et al. use a Rothmann–Keller type model with differently colored sets of distribution functions to simulate fluids with multiple phases. Several other methods exist for multi-phase flows, e.g., [35,40] or [46]. These have also been extended in different ways, e.g., to allow for high density ratios [19]. On the other hand, in [13], Ginzburg et al. present a free surface LBM model that makes use of more complicated boundary conditions, and prescribes shear stresses at the interface. For problems such as rising bubbles, these shear stresses should be considered, but in Sect. 5 we will focus on test cases where they can be neglected.

For the model of this paper, 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

$$\varepsilon(\mathbf{x}, t) = m(\mathbf{x}, t)/\rho(\mathbf{x}, t), \tag{10}$$

where cell density is computed with Eq. (4). For the mass exchange between two interface cells, their fluid fraction is

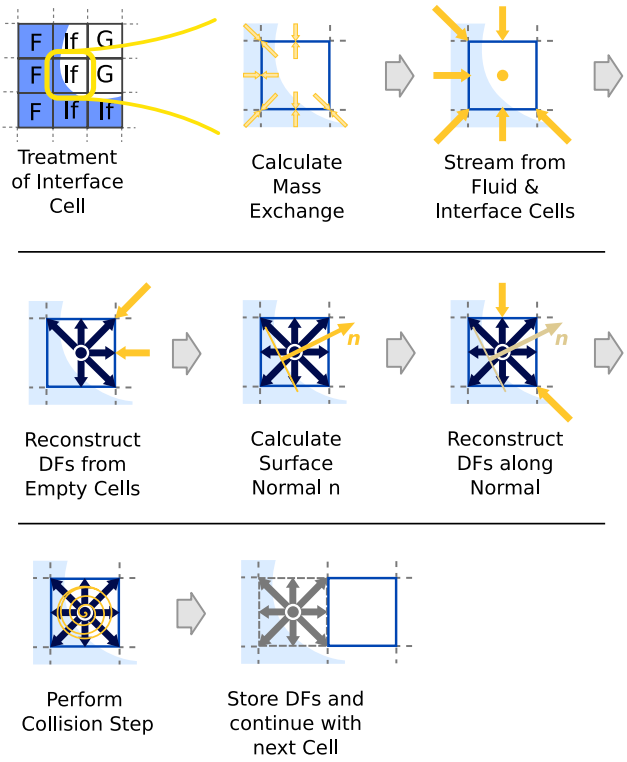

Fig. 5 An illustration of the steps that have to be executed for an interface cell

taken into account to approximate the area they share at the cell boundary:

$$\Delta m_i(\mathbf{x}, t + \Delta t) = \left[f_{\bar{i}}(\mathbf{x} + \Delta t \mathbf{e}_i, t) - f_i(\mathbf{x}, t)\right] \cdot \frac{\varepsilon(\mathbf{x} + \Delta t \mathbf{e}_i, t) + \varepsilon(\mathbf{x}, t)}{2}. \tag{11}$$

In order to guarantee mass conservation, the mass exchange and the computation of the fluid fraction coefficient are symmetric. If the adjacent cell is a fluid cell, the mass exchange is simplified to

$$\Delta m_i(\mathbf{x}, t + \Delta t) = f_{\bar{i}}(\mathbf{x} + \Delta t \mathbf{e}_i, t) - f_i(\mathbf{x}, t) \tag{12}$$

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:

$$m(\mathbf{x}, t + \Delta t) = m(\mathbf{x}, t) + \sum_{i=1}^{19} \Delta m_i(\mathbf{x}, t + \Delta t). \tag{13}$$

For fluid cells, the mass is equal to their density, the fluid fraction being $\varepsilon = 1$. For empty and boundary cells, no mass exchange needs to be considered, as the mass exchange is only computed according to the streaming step, and no DFs are streamed from or into the two latter cell types.

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 $(\mathbf{x} + \Delta t \mathbf{e}_i)$ the post-streaming DF $f_{\bar{i}}'$ is reconstructed as:

$$f_{\bar{i}}'(\mathbf{x}, t + \Delta t) = f_i^{\mathrm{eq}}(\rho_G, \mathbf{u}) + f_{\bar{i}}^{\mathrm{eq}}(\rho_G, \mathbf{u}) - f_i(\mathbf{x}, t). \quad (14)$$

Here $\mathbf{u}$ is the velocity of the interface cell. In this form the boundary conditions do not 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. [41] or [22]) and could thus be combined with the algorithm described in Sect. 3. All DFs that would be streamed from empty cells are calculated with Eq. (14), 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 $\mathbf{n}$ is calculated by finite differences from the fluid fraction values. It is used to determine the velocity directions coming from the gas phase half space given by the surface normal (all f_i with $\mathbf{n} \cdot \mathbf{e}_i < 0$). 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 ($\varepsilon > 1$) or emptied ($\varepsilon < 0$).

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. While previous computations for the boundary conditions and the mass transfer can be computed locally for an interface cell, this conversion handling requires accesses to neighboring cells. 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, thus fluid cells may never have an empty cell neighbor. 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., [43], where interactive simulations were performed, or in [22], where validation experiments including surface tension were evaluated. Overall, in addition to the high computational efficiency, the advantages of the algorithm are the mainly local treatment of the free surface boundary conditions, and the mass conservation up to machine precision. In the following we will demonstrate, that the algorithm can also be used to efficiently produce high quality animations with large grid resolutions.

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 $1/2$. In the following we will apply the Smagorinsky subgrid turbulence model, as used in e.g. [52]. The subgrid model, as derived in [18], models the effect of subgrid scale vortices by modifying the viscosity according to the Reynolds stress tensor, and can be combined with approaches such as MRT [57]. 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 subgrid turbulence model applies the calculation of the local stress tensor as described in [36] to the LBM. This is simplified, since 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 Smagorinsky 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 subgrid vortices [56]. The turbulence model is integrated into the basic algorithm as described in Sect. 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 earlier.

The modified relaxation time τ_s is calculated by performing the steps that are described in the following. First the tensor $\Pi_{\alpha,\beta}$ is obtained for each cell by taking the second moment of the non-equilibrium parts of the distribution functions with

$$\Pi_{\alpha,\beta} = \sum_{i=1}^{19} \mathbf{e}_{i_\alpha} \mathbf{e}_{i_\beta} \left(f_i - f_i^{\mathrm{eq}} \right), \quad (15)$$

where we have used the notation from [18]. Thus α and β each run over the three spatial dimensions, while i is the index of the respective velocity vector for the D3Q19 model.

As in [18], the intensity of the local strain tensor S is then computed as

$$S = \frac{1}{6C^2} \left(\sqrt{\nu^2 + 18C^2 \sqrt{\Pi_{\alpha,\beta} \Pi_{\alpha,\beta}}} - \nu \right). \quad (16)$$

Now the modified relaxation time is computed as

$$\tau_s = 3(\nu + C^2 S) + \frac{1}{2}. \tag{17}$$

From Eq. (16) 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 [44]. 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. The rescaling ensures that dimensionless numbers, such as Reynolds and Froude number, remain the same after the change of the time step. The Mach number, on the other hand, changes due to the rescaling. This is, however, uncritical for free surface flows, such as those presented in Sect. 5. An evaluation of the effects of this Mach number change can be found in [44]. 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 Sect. 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:

$$|\mathbf{u}_{\max}| > \frac{1}{6}/\xi, \quad \text{with} \quad \xi = \frac{4}{5}. \tag{18}$$

We use $1/6$ as the velocity threshold, as it is the half of $1/3$, at which point the equilibrium DFs according to Eq. (3) can become negative. If Eq. (18) holds, the new time step size is given by

$$\Delta t_n = \xi \, \Delta t_o, \tag{19}$$

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 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:

$$\tau_n = s_t \left(\tau_o - \frac{1}{2} \right) + \frac{1}{2}, \quad \text{with} \quad s_t = \Delta t_n / \Delta t_o. \tag{20}$$

The new acceleration for a LBM step is then calculated as

$$\mathbf{g}_n = s_t^2 \mathbf{g}_o. \tag{21}$$

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. (4) for an interface or fluid cell, the new values are computed with:

$$\begin{aligned} \rho_n &= s_t \ (\rho_o - \rho_{\text{med}}) + \rho_{\text{med}} \ \text{ and} \\ \mathbf{u}_n &= s_t \ \mathbf{u}_o, \end{aligned} \tag{22}$$

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:

$$\begin{aligned} m_n &= m_o(\rho_o/\rho_n) \ \text{ and} \\ \varepsilon_n &= m_n/\rho_n, \end{aligned} \tag{23}$$

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 [8]. 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:

$$f_i^n = s_f \ \left[f_i^{\text{eq}}(\rho_o, \mathbf{u}_o) + s_\tau \ \left(f_i - f_i^{\text{eq}}(\rho_o, \mathbf{u}_o) \right) \right], \tag{24}$$

where s_f and s_τ are calculated as follows:

$$\begin{aligned} s_f &= f_i^{\text{eq}}(\rho_n, \mathbf{u}_n) / f_i^{\text{eq}}(\rho_o, \mathbf{u}_o) \\ s_\tau &= s_t(\tau_n/\tau_o). \end{aligned} \tag{25}$$

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 [8], Filippova et al. developed an algorithm to couple LBM simulations of different resolutions. The coupling of

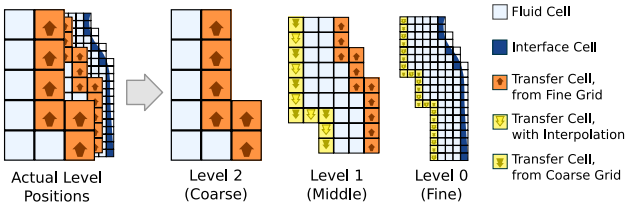

Fig. 6 Example of a coarsened fluid region near the free surface with two levels of coarser grids. To the *left* the transfer cell layers for coupling adjacent levels can be seen

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. (24). 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. E.g., two-phase simulations with MRT using the model from [14] in combination with adaptive grid refinement have been demonstrated by Tölke et al. in [45]. Below, we will use a rescaling similar to the one presented in [45]. Grid refinement has also been used in [54] to compute simulations of an airfoil on a grid with refined blocks. Rohde recently proposed an alternative approach for grid refinement with LBM, see [34] for details. However, since this method requires an additional filtering step to ensure stability, our work is based on the algorithm described in [8].

Figure 6 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. 6, the grid spacing Δx_c and Δt_c on the coarse grid are twice those of the fine grid. According to Eq. (8) this means that the relaxation time needs to be calculated with the corresponding parameters for each grid. Reformulating Eq. (8) using $\Delta x_c = 2\Delta x_f$, the relaxation time for the coarse grid is calculated by

$$\tau_c = \frac{1}{2}\left(\tau_f - \frac{1}{2}\right) + \frac{1}{2}. \tag{26}$$

In Fig. 6 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

$$f_{c,i} = f^{\mathrm{eq}}_{f,i} + s_{cf}\left[f_{f,i} - f^{\mathrm{eq}}_{f,i}\right], \quad \text{with } s_{cf} = \frac{2\tau_c}{\tau_f}. \tag{27}$$

Here we use rescaling factors similar to those proposed in [45], instead of those from [8], as the latter ones have a singularity for $\tau = 1$. For a transfer in the other direction, from the coarse to the fine grid, Eq. (27) becomes

$$f_{f,i} = f^{\mathrm{eq}}_{c,i} + s_{fc}\left[f_{c,i} - f^{\mathrm{eq}}_{c,i}\right], \quad \text{with } s_{fc} = \frac{1}{s_{cf}} = \frac{\tau_f}{2\tau_f}. \tag{28}$$

Note that the rescaling for transfer cells is performed after collision on both grids. Thus, the DFs are only streamed on the fine destination grid, while no collision is necessary, as the DFs are overwritten directly afterwards with DFs from the coarse grid again.

Likewise, fine grid transfer cells (marked with a filled downward arrow) again lie at the same positions as coarse grid cells, thus their DFs are transferred directly with Eq. (27). 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 $f^{\mathrm{eq}}_{c,i}$ of Eq. (27), 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. [54], 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, since 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. 7.

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

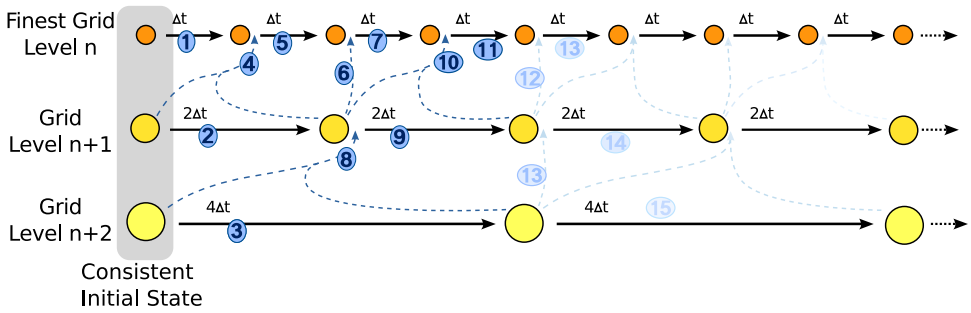

Fig. 7 Here the effect of the different time step sizes for multiple simulation grids is shown. The *numbers* indicate the order in which the steps are performed. *Dashed arrows* indicate interpolation, while *straight arrows* from one *circle* to another represent LBM steps with the indicated time step length

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. Note that this approach also requires a subsequent refinement of initially coarsened regions, once the free surface moves there during the course of the simulation.

3.1 Turbulence model

The free surface extension of Sect. 2.1 and the subgrid model of Sect. 2.2 can be combined directly. The turbulence model differs from approaches such as MRT since it does not change the equilibrium DFs. Furthermore, the free surface equations in Sect. 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 fluid velocities becoming too large during the course of the simulation.

3.2 Adaptive time steps

The adaptive time step procedure from Sect. 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. (20). Instabilities due to τ being almost 0.5 are alleviated by applying the turbulence model. This in turn requires a modification of Eq. (24), as the non-equilibrium scaling of the adaptive time steps depends on the relaxation time τ.

With Eq. (20), the lattice viscosities ν_n and ν_o for the old and the new time step are calculated. Equations (15), (16) and (17) 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. Equation (24) 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

$$s_\tau = s_t(\tau_{s,n} \,/\, \tau_{s,o}). \tag{29}$$

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 Sect. 2.4 cannot be applied without modifications. In [7] and [24] 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. (11) 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 [47], we will denote the fine to coarse grid transfer with *restriction* and the coarse to fine grid transfer with *prolongation* in the following sections.

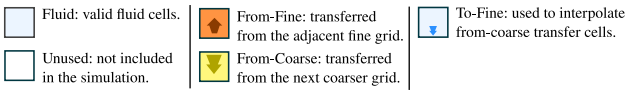

Fig. 8 Cell types for the adaptive coarsening algorithm

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, a cell and its neighborhood, together with the neighborhood of the cell 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 surface 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 Sect. 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 the five cell types shown in Fig. 8:

- *Fluid*: these are valid fluid cells treated as described in Sect. 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 Sect. 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.

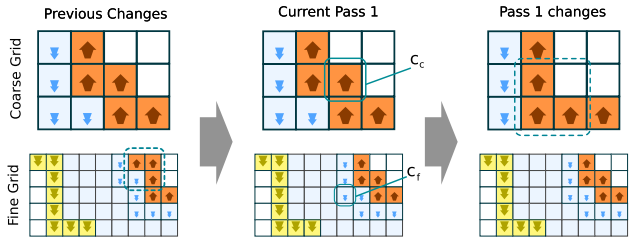

Fig. 9 Pass 1

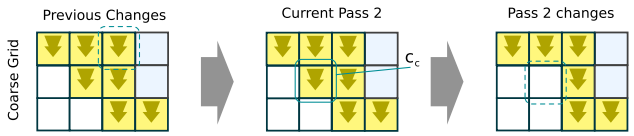

Fig. 10 Pass 2

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 (Fig. 9).

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 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 (Fig. 10).

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 (Fig. 11).

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.

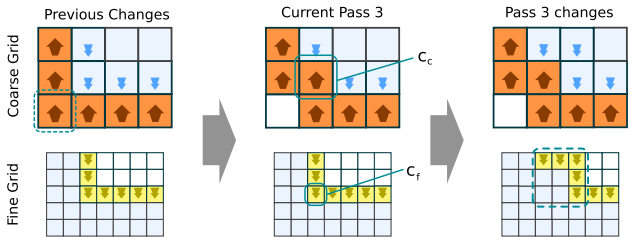

Fig. 11 Pass 3

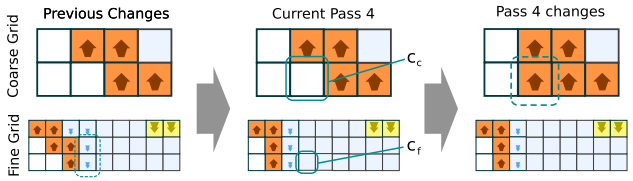

Fig. 12 Pass 4

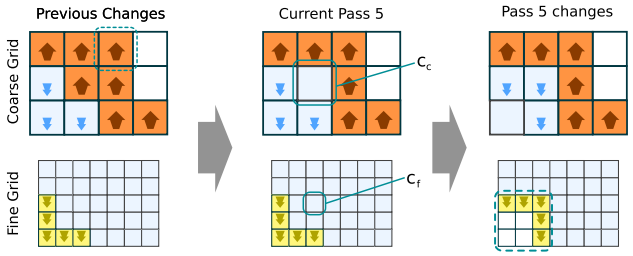

Fig. 13 Pass 5

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 (Fig. 12).

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 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 (Fig. 13).

Although the cell conversion does require five 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 Sect. 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. (27). After interpolation of the DFs, the modified relaxation times τ_{sc} and τ_{sf} are calculated with Eq. (27) using the viscosities ν_c and ν_f, respectively. Finally, the scaling factor s_{cf} is calculated with

$$s_{cf} = \left(\frac{1}{\tau_{sc}} - 1\right) \frac{2\tau_{sc}}{\tau_{sf}}. \tag{30}$$

and used instead of Eq. (27) with Eqs. (27) and (28).

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 Fig. 14. 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 Sect. 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. 14 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

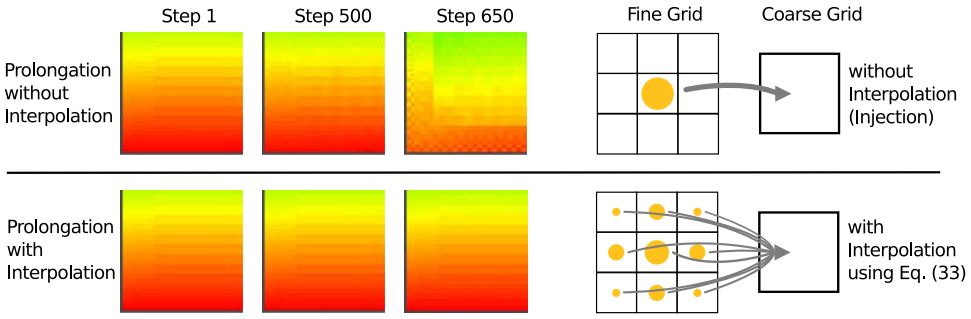

Fig. 14 Example of artifacts that occur for a simple standing fluid test case with a resolution of 128^2 and two coarse levels. Each picture shows the density distribution in the lower left corner of the fluid, where *green values* indicate $\rho = 1.0$ while a *red color* indicates larger values. The *upper row* of pictures was created without any interpolation for the prolongation, while the *lower row* makes use of Eq. (31)

Eq. (28) are calculated as

$$\tilde{f}_{f,i}(\mathbf{x}) = \sum_{j=1}^{19} f_{f,i}(\mathbf{x} + \Delta t \mathbf{e}_j) \frac{w_j}{w_{\text{total}}} \tag{31}$$

with

$$w_j = e^{-|\mathbf{e}_j|} - e^{-2\cdot 3}, \quad w_{\text{total}} = \sum_{j=1}^{19} w_j \tag{32}$$

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

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.

4 Validation

The accuracy of the different grid transfer methods will be determined by comparing $\mathscr{E}$, 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 Figs. 15 and 16 are thus computed as

$$\mathscr{E} = \frac{1}{t_{\text{total}}} \frac{1}{n_{\text{total}}} \sum_{t=1}^{t_{\text{total}}} \sum_{\mathbf{x}\in\Omega} |\varepsilon_{\text{ref}}(\mathbf{x}, t) - \varepsilon(\mathbf{x}, t)|, \tag{33}$$

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 $\mathscr{E}$ is measured at. For Fig. 15 the grid resolution of the reference simulation was used to set the number of measurement points. Note that $\mathscr{E}$ 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 $\nu' = 10^{-6}$ (m^2/s) and an acceleration of $\mathbf{g}' = (0, -9.81, 0)^T$ (m/s^2).

4.1 Test case with static coarsening

The different interpolation methods will be tested with a setup of a drop falling into a standing fluid, similar to Figs. 17, 18, 19. The lower half of the domain is statically coarsened. During the course of the simulation the free surface keeps a distance of several cells to the coarsened region, hence the coarse cells grids do not have to be changed. Figure 15 shows results in two and three dimensions, to the left and right, respectively, each for three grid resolutions. The

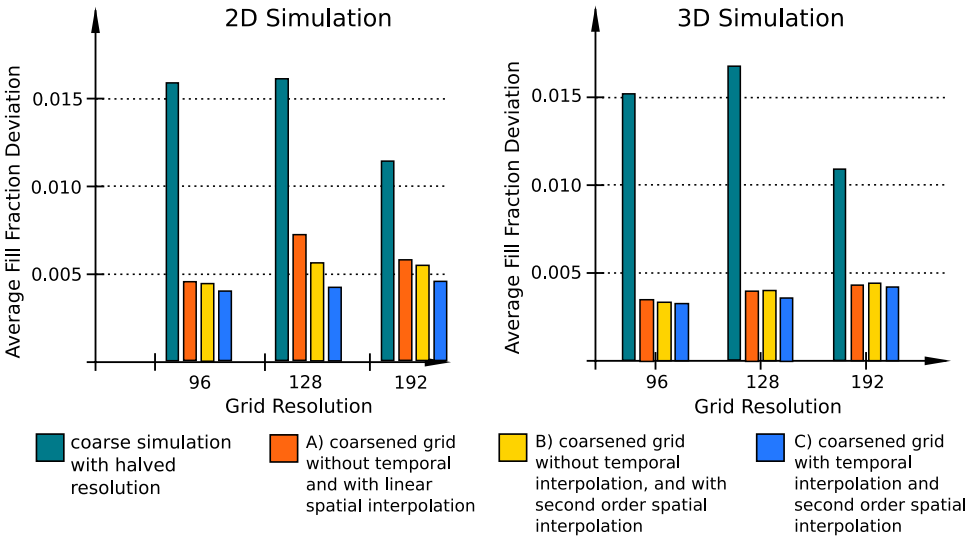

Fig. 15 Accuracy measurement for the interpolation test case with static coarsening

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 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 [55], 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 [31] 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

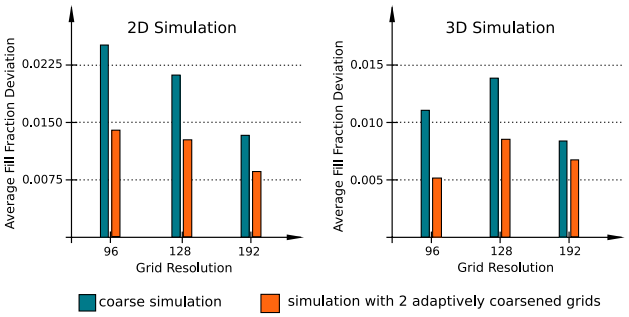

Fig. 16 Accuracy measurement for the dynamic test case with the adaptive coarsening

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 Test case with dynamic coarsening

To validate the accuracy of the adaptive coarsening technique described in Sect. 3 we have used a breaking liquid column setup similar to Fig. 20. 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 back and forth, which makes constant updates of the coarsened region necessary.

Accuracy measurements of $\mathscr{E}$ computed with Eq. (33) are shown in Fig. 16. Here again a coarse simulation with half

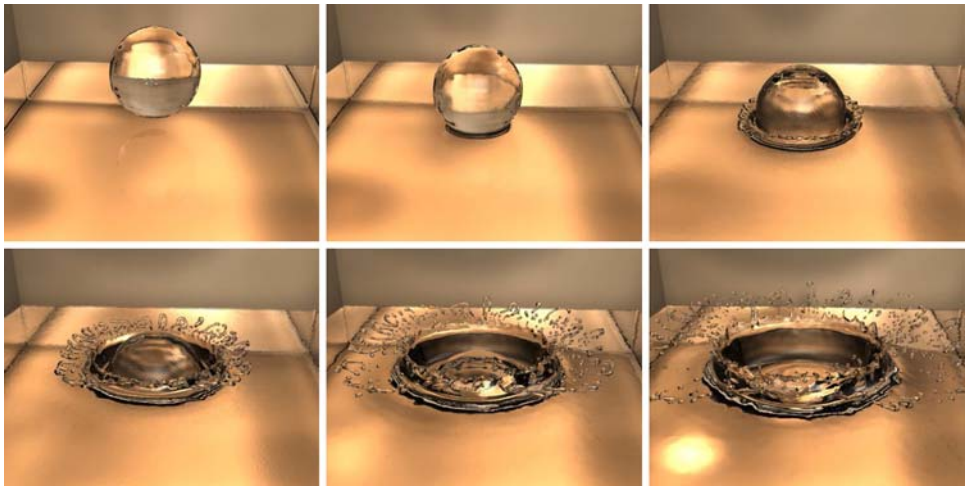

Fig. 17 Images of the falling drop simulation with a grid resolution of 480^3 and the adaptive coarsening algorithm. The simulation was parametrized to represent a basin of water with 10cm side length

Table 1 Workload distribution for an typical simulation

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

Table 2 Performance measurements of the basic free surface simulation code without adaptive coarsening on different architectures with up to four processors

CPU	MLSUPS
Pentium4 3.2 GHz	1.84
Athlon64 2.4 GHz	1.98
4-way Opteron 4 · 2.2 GHz (with OpenMP)	3.73

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 several examples of detailed simulations and illustrate the speedup that can be achieved by adaptive coarsening.

5 Results and 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 and three coarse levels.

As can be seen in Table 1, the majority of the computations is 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 updates per second: *MLSUPS* (million lattice site updates per second). However, this is not valid anymore once adaptive grid resolutions are involved. In this case, it is crucial how much faster the overall simulation is done in comparison to a standard simulation using a single grid level. The following tables show 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 different 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 would still be low. In the following we will demonstrate the achievable speedups with the test cases shown in

- Figure 17(A), the impingement of a falling drop on a fluid surface, and
- Figure 20(B), the breaking of a column of liquid.

Both cases were run in two different sizes: 120^3 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

Table 3 Reynolds number (Re) and Froude number (Fr) for the three test cases used in Sect. 5. The domain size is 0.1m, $\nu' = 10^{-6}$ [m^2/s] and $\mathbf{g}' = (0, -9.81, 0)^T$ [m/s^2]

	Falling drop	Breaking dam	Filled glass
Re	200,000	450,000	100,000
Fr	14.28	5.32	6.39
L [m]	0.02	0.09	0.025
$\mathbf{v}$ [m/s]	10	5	4
H [m]	0.05	0.09	0.04

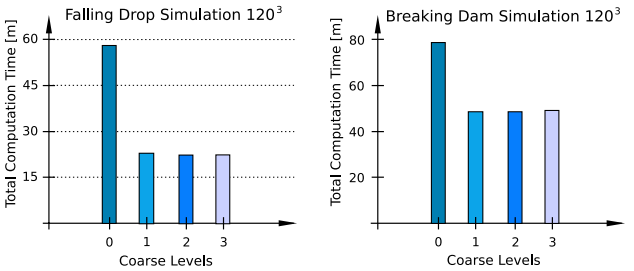

Fig. 18 Performance for a resolution of 120^3 on a single Pentium4 CPU with 3.2 GHz

left is run only on the finest level, while the others use up to three levels of adaptive coarsening.

Two dimensionless variables, the Reynolds number (Re) and the Froude number (Fr) for the three test cases are shown in Table 3. The Reynolds number represents the ratio between a characteristic length L times velocity $\mathbf{v}$ and the viscosity ($Re = L\mathbf{v}/\nu$), while the Froude number relates the velocity to the gravity and water height H ($Fr = \mathbf{v}/\sqrt{\mathbf{g}H}$). Note that the high Reynolds numbers are the result of the chosen viscosity of water, which is close to zero. These parametrizations clearly represent the upper limits of this method. As the primary goal of these test cases was the measurement of the performance of the adaptive coarsening, these parameters were chosen to test the limits of stability of the algorithm.

In Fig. 18 the performance for the relatively small resolution of 120^3 on a Pentium4 CPU with 3.2 GHz is visible. For test case A, speedup 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.

The performance results of Fig. 19 are for a resolution of 480^3 on a four-way Opteron node with 2.2 GHz for each of the four CPUs. The traversal of the finest grid was parallelized with OpenMP. As was demonstrated above, the majority

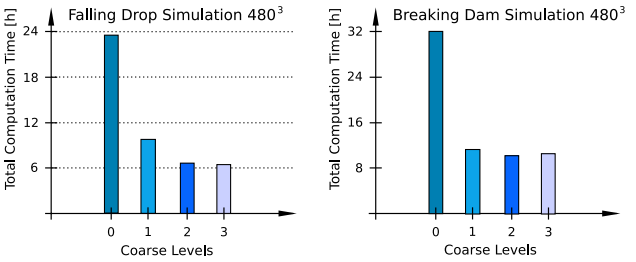

Fig. 19 Performance with OpenMP parallelization for a resolution of 480^3 on a four-way Opteron node (each CPU with 2.2 GHz)

of the work is done on the finest grid—thus the parallelization is only applied to the traversal of the finest grid level. 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 contrast to the 120^3 runs, more than a single level of coarsening yields a further speedup for test case A.

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 [48]. It allows the productions of high quality fluid animations as shown in Fig. 21, and was used to produce the visualizations of our simulation runs. The sources for the solver including the implementation of our adaptive coarsening algorithm were released under the *GNU Public License*, and are available on the same website.

6 Conclusions and outlook

We have presented a stable method for free surface simulations with the LBM. It can be used to efficiently perform simulations with large volumes of fluid and thus enables the creation of highly detailed and physically correct fluid animations. This is achieved by our algorithm to adaptively coarsen the simulation resolution inside of larger 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 subgrid turbulence model and an adaptive time step algorithm ensures stability of the fluid simulator. We have validated the algorithm by 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, speedup 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

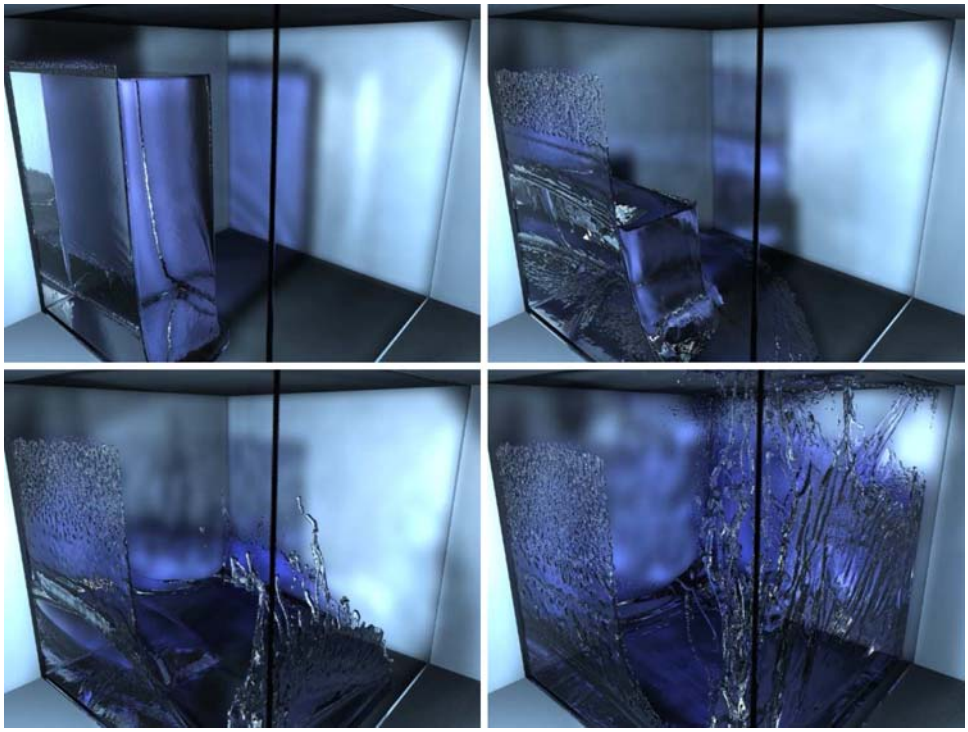

Fig. 20 Pictures of the breaking dam setup, again with a grid resolution of 480^3 and a parametrization of a 10cm domain with water

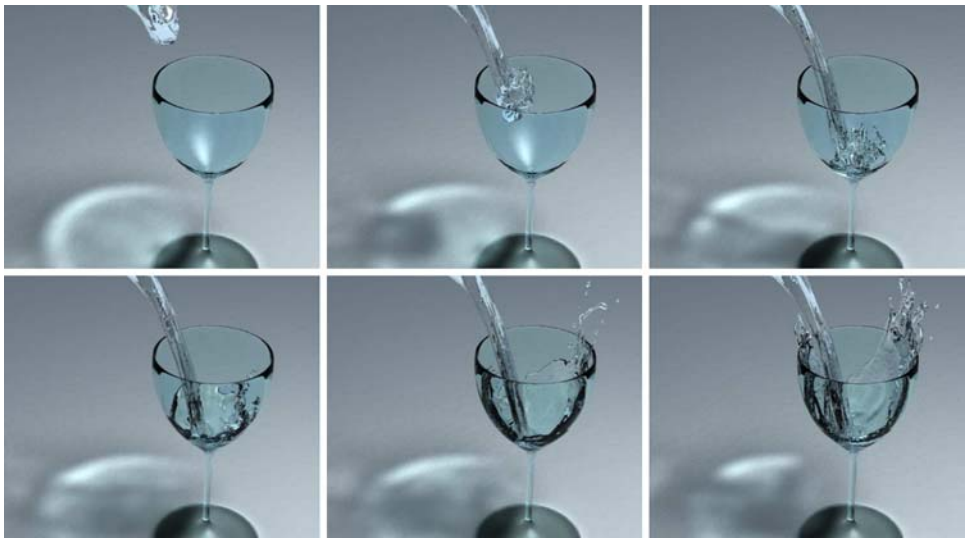

Fig. 21 Several frames of animation from a simulation of filling a glass shaped obstacle

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.

In order to e.g. accurately resolve near wall shear layers of turbulent flows, the coarsening criterion could be changed to only coarsen areas with a low shear stress. Another challenge will be to efficiently parallelize our algorithm to run on large distributed memory systems, as was e.g. demonstrated for adaptive multigrid solvers in [1]. Such a parallelization becomes more difficult once adaptive approaches, like the one presented here, are used, since the information across all grids involved in the simulation needs to be synchronized. In order to further enhance the impression of large fluid scenes for animations, the three dimensional simulation could be coupled to a two dimensional one for efficient simulations of large water surfaces.

As the motion of the free surface is naturally hard to estimate from still pictures, we have made the animations corresponding to Figs. 17, 20 and 21 available on our website [42].

Acknowledgments This research is funded by the DFG Graduate College GRK-244 *3-D Image Analysis and Synthesis*. We furthermore thank Carolin Körner for the metal foam samples, and Thomas Zeiser for the helpful discussions.

References

1. Bastian, P., Birken, K., Lang, S., Johannsen, K., Neuß, N., Rentz-Reichert, H., Wieners, C.: UG: A flexible software toolbox for solving partial differential equations. Comput. Vis. Sci. **1**, 27–40 (1997)
2. Bhatnagar, P.L., Gross, E.P., Krook, M.: A model for collision processes in gases. Phys. Rev. **94**, 511–525 (1954)
3. Bouzidi, M., Firadouss, M., Lallemand, P.: Momentum transfer of a lattice-boltzmann fluid with boundaries. Phys. Fluids **13**, 3452–3459 (2002)
4. Buwa, V.V., Deo, D.S., Ranade, V.V.: Eulerian–Lagrangian simulations of unsteady gas–liquid flows in bubble Columns. Int. J. Multiphase Flow (2005)
5. Causin, P., Miglio, E., Saleri, F.: Algebraic factorizations for 3D non-hydrostatic free surface flows. Comput. Vis. Sci. **5**(2), 85–94 (2002)
6. Chu, N.S.H., Tai, C.L.: MoXi: real-time ink dispersion in absorbent paper. ACM Trans. Graph. **24**(3), 504–511 (2005)
7. Crouse, B., Krafcyzk, M., Tölke, J., Rank, E.: A LB-based approach for adaptive flow simulations. Int. J. Modern Phys. B **17**, 109–112 (2003)
8. Filippova, O., Hänel, D.: Grid refinement for lattice-BGK models. J. Comp. Phys. **147**, 219–228 (1998)
9. Frisch, U., d'Humières, D., Hasslacher, B., Lallemand, P., Pomeau, Y., Rivert, J.P.: Lattice gas hydrodynamics in two and three dimensions. Complex Syst. **1**, 649–707 (1987)
10. Geist, R., Rasche, K., Westall, J., Schalkoff, R.: Lattice-Boltzmann Lighting. In: Proceedings of Eurographics Symposium on Rendering 2004, pp. 355–362 (2004)
11. Geller, S., Krafczyk, M., Tölke, J., Turek, S., Hron, J.: Benchmark computations based on Lattice-Boltzmann, Finite Element and Finite Volume Methods for laminar Flows. Comput. Fluids **35**, 8–9 (2006)
12. Ginzburg, I., d'Humières, D.: Multi-reflection boundary conditions for lattice Boltzmann models. Phys. Rev. E **68:066614-1-30** (2003)
13. Ginzburg, I., Steiner, K.: Lattice Boltzmann model for free-surface flow and its application to filling process in casting. J. Comp. Phys. **185/1** (2003)
14. Gunstensen, A.K., Rothman, D.H., Zaleski, S., Zanetti, G.: Lattice Boltzmann model of immiscible fluids. Phys. Rev. A **43** (1991)
15. He, X., Luo, L.S.: A priori derivation of lattice Boltzmann equation. Phys. Rev. E **55**, R6333–R6336 (1997)
16. He, X., Luo, L.S.: Lattice Boltzmann model for the incompressible Navier–Stokes equations. J. Stat. Phys. **88**, 927–944 (1997)
17. Hirt, C.W., Nichols, B.D.: Volume of fluid (VOF) method for the dynamics of free boundaries. J. Comp. Phys. **39**, 201–225 (1981)
18. Hou, S., Sterling, J.D., Chen, S., Doolen, G.: A lattice Boltzmann subgrid model for high Reynolds number flow. Fields Inst. Commun. **6**, 151–166 (1996)
19. Inamuro, T., Ogata, T., Tajima, S., Konishi, N.: A lattice boltzmann method for incompressible two-phase flows with large density differences. J. Comp. Phys. **198**, 628–644 (2004)
20. Körner, C., Pohl, T., Rüde, U., Thürey, N., Zeiser, T.: Parallel lattice Boltzmann methods for CFD applications. In: Bruaset, A., Tveito, A. (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, LNCSE, vol. 51, pp. 439–465. Springer, Heidelberg (2005)
21. Körner, C., Singer, R.: Numerical Simulation of Foam Formation and Evolution with Modified Cellular Automata. Metal Foams and Porous Metal Structures, pp. 91–96 (1999)
22. Körner, C., Thies, M., Hofmann, T., Thürey, N., Rüde, U.: Lattice Boltzmann model for free surface flow for modeling foaming. J. Stat. Phys. **121**(1-2), 179–196 (2005)
23. Körner, C., Thies, M., Singer, R.F.: Modeling of metal foaming with lattice Boltzmann automata. Adv. Eng. Mater. (2002)
24. Krafczyk, M.: Gitter–Boltzmann-Methoden, von der Theorie zur Anwendung. Habilitation (2001)
25. Lallemand, P., Luo, L.S.: Theory of the lattice Boltzmann method: dispersion, dissipation, isotropy, Galilean invariance, and stability. Phys. Rev. E **61**(6), 6546–6562 (2000)
26. Losasso, F., Gibou, F., Fedkiw, R.: Simulating water and smoke with an octree data structure. ACM Trans. Graph. **23**(3), 457–462 (2004)
27. Krafczyk, M., Tölke, J., Rank, E., Schulz, M.: Two-dimensional simulation of fluid-structure interaction using lattice-Boltzmann methods. Comput. Struct. **79** (2001)
28. Mei, R., Luo, L.S., Shyy, W.: An accurate curved boundary treatment in the lattice Boltzmann method. J. Comp. Phys. **155**, 307–330 (1999)
29. Monaghan, J.: Smoothed particle hydrodynamics. Ann. Rev. Astron. Phys. **30**, 543–574 (1992)
30. Müller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation, pp. 154–159 (2003)
31. Pohl, T., Kowarschik, M., Wilke, J., Iglberger, K., Rüde, U.: Optimization and Profiling of the Cache Performance of Parallel Lattice Boltzmann Codes in 2D and 3D. Technical Report 3–8, Germany (2003)
32. Pohl, T., Thürey, N., Deserno, F., Rüde, U., Lammers, P., Wellein, G., Zeiser, T.: Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures. In: Proceedings of supercomputing conference 2004 (2004)
33. Qian, Y.H., d'Humières, D., Lallemand, P.: Lattice BGK models for Navier–Stokes equation. Europhys. Lett. **17**(6), 479–484 (1992)
34. Rohde, M., Kandhai, D., Derksen, J.J., van den Akker, H.E.A.: A generic mass conservative local grid refinement technique for lattice-Boltzmann schemes. Int. J. Num. Methods Fluids **51**, 439 (2006)
35. Shan, X., Chen, H.: Simulation of non-ideal gases and liquid–gas phase transitiions by the lattice Boltzmann equation. Phys. Rev. E **49**, 2941–2948 (1994)
36. Smagorinsky, J.: General circulation experiments with the primitive equations. Mon. Wea. Rev. **91**, 99–164 (1963)
37. Stam, J.: Stable Fluids. In: Proceedings of ACM SIGGRAPH, pp. 121–128 (1999)
38. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, Oxford (2001)
39. Sussman, M.: 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)
40. Swift, M.R., Orlandi, E., Osborn, W.R., Yeomans, J.M.: Lattice Boltzmann simulations of liquid–gas and binary fluid systems. Phys. Rev. E **54**, 5041–5052 (1996)
41. Thuerey, N.: A lattice Boltzmann method for single-phase free surface flows in 3D. Masters Thesis, Department of Computer Science 10, System-Simulation, University of Erlangen-Nuremberg (2003)

42. Thürey, N., Rüde, U.: Webpage: stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids. http://www10.informatik.uni-erlangen.de/~sinithue/sfsflbmacg/
43. Thürey, N., Körner, C., Rüde, U.: Interactive free surface fluids with the lattice Boltzmann method, Technical Report 05–4. Technical Report, Department of Computer Science 10, System Simulation (2005)
44. Thürey, N., Pohl, T., Rüde, U., Oechsner, M., Körner, C.: Optimization and stabilization of LBM free surface flow simulations using adaptive parameterization. Comput. Fluids **35**(8–9), 934–939 (2006)
45. Tölke, J., Freudiger, S., Krafcyzk, M.: An adaptive scheme using hierarchical grids for lattice Boltzmann multi-phase flow simulations. Comput. Fluids **17**, 109–112 (2003)
46. Tölke, J., Krafcyzk, M., Schulz, M., Rank, E.: Lattice Boltzmann simulations of binary fluid flow through porous media. Philos. Trans. R. Soc. Lond. A **360**, 535–545 (2002)
47. Trottenberg, U., Oosterlee, C., Schüller, A.: Multigrid. Academic Press, London (2001)
48. Veldhuizen, B., Langlotz, J., et al.: Blender open source 3D graphics creation (2005) http://www.blender3d.org
49. Verberg, R., Ladd, A.J.C.: Accuracy and stability of a lattice-boltzmann model with subgrid scale boundary conditions. Phys. Rev. E **65**(016701-1-6) (2001)
50. Wang, C., Wang, Z., Xia, T., Peng, Q.: Real-time snowing simulation. The Visual Computer, pp. 315–323 (2006)
51. Wei, X., Li, W., M"uller, K., Kaufman, A.E.: The lattice-Boltzmann method for simulating gaseous phenomena. IEEE Trans. Vis. Comput. Graph. **10**(2), 164–176 (2004)
52. Wei, X., Zhao, Y., Fan, Z., Li, W., Yoakum-Stover, S., Kaufman, A.: Natural phenomena: blowing in the wind. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation, pp. 75–85 (2003)
53. Wittum, G.: Multi-grid methods for Stokes and Navier–Stokes equations with transforming smoothers: algorithms and numerical results. Numer. Math. **54**, 543–563 (1989)
54. Yu, D., Mei, R., Luo, L.S., Shyy, W.: Viscous flow computations with the method of lattice Boltzmann equation. Prog. Aerospace Sci. **39**(5) (2003)
55. Yu, D., Mei, R., Shyy, W.: A multi-block lattice Boltzmann method for viscous fluid flows. Int. J. Numer. Methods Fluids **39** (2002)
56. Yu, H., Girimaji, S., Luo, L.S.: Lattice Boltzmann simulations of decaying homogeneous isotropic turbulence. Phys. Rev. E **71** (2005)
57. Yu, H., Luo, L.S., Girimaji, S.: LES of turbulent square jet flow using an MRT lattice Boltzmann model. Comput. Fluids **25**, 957–965 (2006)