# Interactive Free Surface Fluids with the Lattice Boltzmann Method

**Article** · January 2005

CITATIONS
54

READS
1,069

**3 authors**, including:

Ulrich Rüde
Friedrich-Alexander-University Erlangen-Nürnberg
**613** PUBLICATIONS **8,582** CITATIONS
SEE PROFILE

Carolin Körner
Friedrich-Alexander-University Erlangen-Nürnberg
**345** PUBLICATIONS **15,383** CITATIONS
SEE PROFILE

**FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG**
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

**Lehrstuhl für Informatik 10 (Systemsimulation)**

# Interactive Free Surface Fluids with the Lattice Boltzmann Method

N. Thürey, U. Rüde, C. Körner

Technical Report 05-4

# Interactive Free Surface Fluids with the Lattice Boltzmann Method

Nils Thürey[1], Ulrich Rüde[1] and Carolin Körner[2]

[1] Institute for System-Simulation, [2] Insititue of Science and Technology of Metals,
University of Erlangen-Nuremberg, Germany

**Abstract**

*In this paper we present our algorithm for animating fluids with a free surface. It is based on the Lattice-Boltzmann Method, instead of a direct discretization of the Navier-Stokes equations. This allows a relatively simple treatment of the free surface boundary conditions at high computational efficiency, without sacrificing the underlying physics. We give a detailed description of our algorithm, focussing on details that are required to achieve a good visual appearance. Furthermore we describe how to implement our adaptive time stepping technique to achieve flexible and stable simulations. We will demonstrate the speed and capabilities of the method with animations from different interactive test cases. These run with, on average, more than 20 frames per second on a standard desktop PC.*

## 1. Introduction

As fluids with a free surface are part of our every day life their believable animation is crucial for any virtual environment that tries to achieve a realistic appearance. Despite of the high complexity, there is a wide selection of work enabling the production of high quality visualizations of fluids for e.g. raytracing. [FAMO99] and [EMF02], among others, have established variants of the level set algorithm to track the free surface, while computing the underlying fluid motion with a discretization of the Navier-Stokes equations. Although these methods are stable, flexible and realistically simulate complex flows, the algorithms require many steps and usually a high number of particles to correctly trace the fluid interface and conserve mass. Still, their success poses the challenge to introduce free surface fluids into real-time applications like games or virtual reality environments.

## 2. Previous Work

Up to now, the NS solver proposed e.g. in [Sta03] has been used to compute flows such as smoke in real-time. [WZF*03] have also used the Lattice Boltzmann Method (*LBM*) to compute interactive flows using graphics hardware. They achieved an impressive number of cell updates per second, however, only for flows without a free surface. [MCG03] proposed smoothed particle hydrodynamics to interactively compute the motion of a free surface fluid. Although they achieved good results, the particles representing the fluid require irregular memory accesses due to the changing neighborhoods of the particles, and a high number of these is required to get a smooth representation of the fluid surface. The triangulation of the fluid surface furthermore can not be directly computed from the particles, and thus requires additional work.

The free surface algorithm presented here uses the LBM, explicitly conserves mass up to machine precision and includes the tracking of the fluid surface. The surface tracking is similar to *Volume of Fluid* (VOF) methods for NS solvers [HN81], which can be used to achieve results of high quality, as e.g. shown in [Sus03]. Though, in contrast to the standard VOF methods, the algorithm presented here directly computes the mass changes from the values available in the LBM. Other multiphase and free surface models for LBM such as [GRZZ91] and [GS03] exist, but the boundary conditions presented in the following are inexpensive to compute. Thus, the algorithm achieves a high performance on common PC architectures as it furthermore requires no particles to be traced. The regularity of the cell array results in a high cache efficiency, thus overcoming the memory bottleneck that often limits the performance [PKW*03]. The algorithm performs very well in parallelized versions, as was shown in e.g. [PTD*04], and thus will furthermore benefit from CPUs with multiple cores.

The algorithm was originally developed for the simulation of metal foams to optimize and enhance the production process [KS00]. The liquid metal behaves almost like water, and the huge areas of fluid gas interface required an efficient and accurate algorithm to compute the development of the foam. [KTS02] produced first results in two dimensions validating the physical correctness of the free surface boundary conditions. In [Thü03] and [TR04] the algorithm was then extended to three dimensions. There it's applicability to the animation of free surface flows was demonstrated by several larger test cases. We will show that with the modifications presented here, the algorithm is not only interesting for technical applications or raytracing, but also for a wide variety of the afore mentioned real-time applications.

## 3. Why Lattice Boltzmann?

The LBM can be imagined as a type of cellular automaton – the simulation region is divided into a cartesian (and in our case, equidistant) grid of cells, each of which only interacts with cells in its direct neighborhood. An overview of the method can also be found e.g. in [Suc01]. While conventional solvers directly discretize the Navier-Stokes (*NS*) equations, the LBM is essentially a first order explicit discretization of the Boltzmann equation in a discrete phase-space, which describes all molecules with their corresponding velocities. The LBM evolved from methods for the simulation of gases, that computed the motion of each

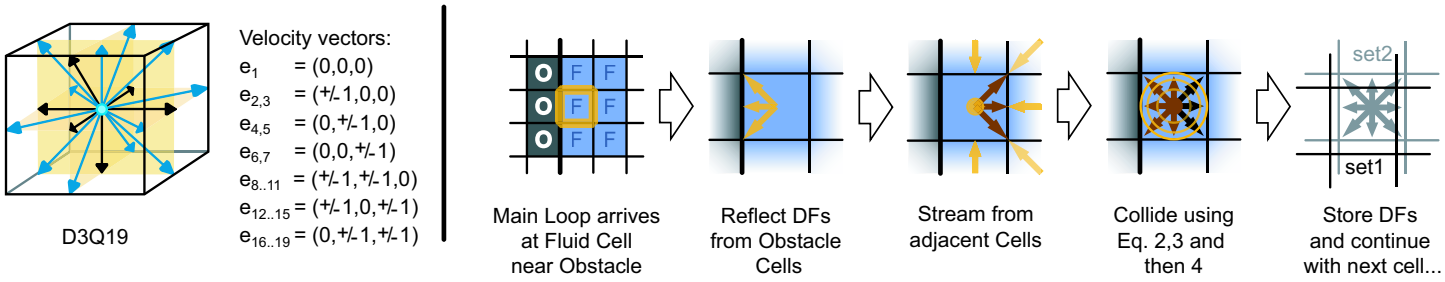


**Figure 1:** *To the left the D3Q19 model with its 19 velocities can be seen. To the right, the D2Q9 model is used to give an overview over the steps of the basic LBM algorithm for a fluid cell next to a vertical obstacle. Note that for clarity the velocity vectors shown in this and the following figures are drawn with half their absolute length.*

molecule in the gas purely with integer operations. For the LBM larger volumes of the fluid are averaged, and the movement and collision of these particle ensembles through the grid is computed, resulting in an accurate reproduction of the NS equations. This can be shown in two different ways – either by Chapman-Enskog expansion from statistical physics [FdH*87], or by direct discretization of the Boltzmann equation [HL97b].

A general comparison of LBM solvers and conventional NS solvers is difficult, however [GKT*04] recently compared state of the art solvers of both kinds. They came to the conclusion, that there is no clear winner, but the LBM even performs better for some problems. Generally, a simple LBM implementation performs very well for complex geometries. As each LBM cell contains information not only about the fluid velocity and pressure, but also about their spatial derivatives, the method allows a very accurate representation of obstacles even for coarse grids. The free surface of a fluid often results in complex and moreover time dependent topologies. This was the motivation for the development of the method presented here, which is especially simple due to the ability of LBM to model complex boundary conditions. Nowadays the LBM is available in commercial fluid solvers [LLS00], which are in production use of e.g. aerospace or car companies.

The underlying base algorithm – the standard LBM – consists of two steps, the stream- and the collide-step, usually combined with no-slip boundary conditions for the domain boundaries or obstacles. The simplicity of the algorithm shows during implementation, which for the base algorithm requires roughly a single page of C-code. Using LBM the particle movement is restricted to a very limited number of directions. We use a three dimensional model with 19 velocities (commonly denoted as *D3Q19*), but there also exist models with 15 or 27. The most common model for two dimensions is D2Q9 with nine velocities. The D3Q19 model with its velocity vectors $\mathbf{e}_{1..19}$ is shown in Figure 1 , together with an overview of the basic LBM algorithm, using the D2Q9 model for clarity. As all formulas for LBM usually only depend on the so-called particle distribution functions (*DFs*), all of these two and three dimensional models can be used with the method presented here. However, for three dimensions D3Q19 is the preferable model.

For each of the velocities a floating point number ($f_{1..19}$) representing the fraction of particles moving with this velocity needs to be stored. For simplicity, the size of a cell $\Delta x$ and the size of a time step $\Delta t$ both are normalized to 1. Thus in the D3Q19 model there are particles not moving at all ($f_1$), moving with speed 1 ($f_{2..7}$) and moving with speed $\sqrt{2}$ ($f_{8..19}$). Note that the order of specifying DFs and velocity vectors used here is not the only possibility. As long as both DFs and velocities use the same, any permutation is valid. In the following, a subscript of $\tilde{i}$ will denote the value from the inverse direction of a value with subscript $i$. Thus $f_i$ and $f_{\tilde{i}}$ are opposite DFs with inverse velocity vectors $\mathbf{e}_{\tilde{i}} = -\mathbf{e}_i$.

During the first part of the algorithm (the stream step) all DFs are advected with their respective velocities. Due to the normalized time step and cell size this results in a movement of the floating point values to the neighboring cells, as shown in Figure 1. Formulated in terms of DFs the stream step can be written as:

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

These post-streaming DFs $f'_i$ have to be distinguished from the standard DFs $f_i$, and are never really stored in the grid. The stream step alone is clearly not enough to simulate the behavior of incompressible fluids, which is governed by the ongoing collisions of the particles among each other. The second part of the LBM, the collide step, amounts for this by weighting the DFs of a cell with the so called *equilibrium distribution functions*, denoted by $f^{eq}$. These depend solely on the density and velocity of the fluid. Here we use the incompressible model from [HL97a]. The density and velocity can be computed by summation of all the DFs for one cell:

$$\rho = \sum f_i \qquad \mathbf{u} = \sum \mathbf{e}_i f_i \quad . \tag{2}$$

Now for a single direction $i$, the equilibrium DF $f_i^{eq}$ can be computed with:

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

The equilibrium DFs represent the state of the fluid where during each time step the amount of particles that are pushed out of each discrete direction by collision is equal to the amount that is pushed into each direction again. This, however, does not mean the fluid is not moving, only that the values of the DFs would not change, if the whole fluid was at equilibrium state similar to a Stokes flow. The collisions of the molecules in a real fluid are approximated by linearly relaxing the DFs of a cell towards their equilibrium state. Thus, each $f_i$ is weighted with the corresponding $f_i^{eq}$ using:

$$f_i(\mathbf{x}, t+\Delta t) = (1-\omega) f'_i(\mathbf{x}, t+\Delta t) + \omega f_i^{eq}. \tag{4}$$

Here $\omega$ is the parameter that controls the viscosity of the fluid. It has to be in the range of 0..2, where values close to 0 result in very viscous fluids, while values near 2 result in more turbulent flows. Usually these are also visually more interesting, however for values close to 2 the method can become instable when high velocities occur. To enhance the

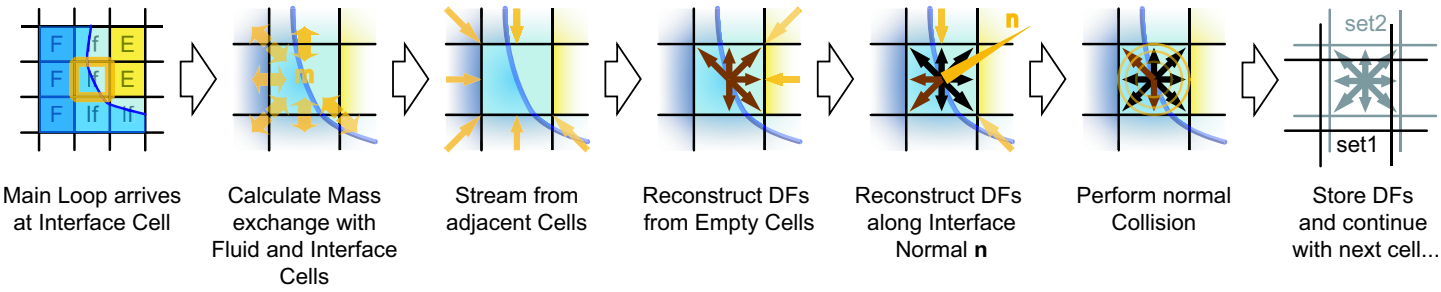


**Figure 2:** *Here an overview over the steps that have to be executed for an exemplary interface cell is given.*

stability of the algorithm, so called *multi relaxation time* (MRT) models [LL00] could be used for the collision. However, we have found the model described here to be sufficient. The parameter $\omega$ relates to the kinematic viscosity of a fluid $\nu$ given in units of the LBM lattice with $\nu = (1/\omega - 0.5)$. The values computed with Equation 4 are stored as DFs for time $t + \Delta t$. As each cell needs the DFs of the adjacent cells from the previous time step, usually two arrays for the DFs of the current and the last time step are used.

The easiest way to implement the no-slip boundary conditions is the bounce back rule, that results in a placement of the boundary in the middle of fluid and obstacle cells. If the neighboring cell at $(\mathbf{x} + \mathbf{e}_i)$ is an obstacle cell during streaming, the DF from the inverse direction of the current cell is used. Thus Equation 1 changes to:

$$f'_{\bar{i}}(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x}, t). \tag{5}$$

Hence, an implementation of the algorithm described so far might consist of a flag field to distinguish fluid and obstacle cells, and an array $DF[2][size_x][size_y][size_z][19]$ of single-precision floating point variables. During a loop over all cells in the grid, each cell collects the neighboring DFs according to Equation 1 or Equation 5, for adjacent fluid and obstacle cells respectively. The density and velocity are computed and used to compute the equilibrium DFs which are then weighted with the streamed DFs and written into the other array, continuing with the next cell in the grid. Subsequent time steps alternate in streaming and colliding the DFs from one array to the other. Note that using Equation 5 the DFs for obstacle cells are never touched.

Looking at the algorithm so far it can be noted that in comparison with a simple finite-difference NS solver, the implementation is much simpler, however, requires more memory. An NS solver would only require $(size_x * size_y * size_z * 7)$ floating point values, but for some cases might need higher resolutions to resolve obstacles with the same accuracy. Using a more sophisticated LBM implementation, the memory requirements could be reduced to $((size_x + 1) * (size_y + 1) * (size_z + 1) * 19)$. Furthermore, note that an adaptive time step size is common practice for NS solver, while the size of the time step in the LBM is by default fixed to 1. In Section 5 we explain how to adaptively resize the time step for LBM [TPR*05]. As the maximum velocity may not exceed $1/3$ in order for the LBM to remain stable, it might still need several time steps to advance to the same time an NS solver would reach in a single step. However each of these time steps usually requires much less work, as the LBM can be computed very efficiently on modern CPUs, and does not require additional computations such as the pressure correction step.

## 4. Free surfaces

The simulation of free surfaces clearly requires a distinction between regions that contain fluid and regions that don't. This is done by marking cells that contain no fluid as empty in the flag field. As with obstacle cells, the DFs of these cells can be completely ignored during the simulation, however, in contrast to boundary cells, the fluid might at some point in the simulation move into this area. To track the fluid motion, another cell type is introduced: the interface cell. These cells form a closed layer between fluid and empty cells. Here the real work for the simulation and tracking of the free surface is done. It consists of three steps – the computation of the interface movement, the boundary conditions at the fluid interface, and the re-initialization of the cell types. In the following, the steps that are executed for an interface cell instead of the standard stream and collide from the previous section are described. On overview of the procedure is given in Figure 3 .

### 4.1. Interface movement

The movement of the fluid interface is tracked by the calculation of the mass that is contained in each cell. For this two additional values need to be stored for each cell, the mass $m$ and the fluid fraction $\varepsilon$. The fluid fraction can be computed with the mass by dividing it through the density of the cell ($\varepsilon = m/\rho$). Similar to VOF method, the interface motion if track by computing the the fluxes between the cells. However, as the DFs correspond to a certain number of particles, the change of mass can be directly computed from the values that are streamed between two adjacent cells for each of the directions in the model. For an interface cell and a fluid cell at $(\mathbf{x} + \mathbf{e}_i)$ this is simply

$$\Delta m_i(\mathbf{x}, t + \Delta t) = f_{\bar{i}}(\mathbf{x} + \mathbf{e}_i, t) - f_i(\mathbf{x}, t). \tag{6}$$

The first DF is the amount of fluid that is entering this cell in the current time step, the second one the amount that is leaving the cell. The mass exchange for two interface cells has to take into account the area of the fluid interface between the two cells. This is approximated by averaging the fluid fraction values of the two cells. Thus Equation 6 becomes

$$\Delta m_i(\mathbf{x}, t + \Delta t) = \left(f_{\bar{i}}(\mathbf{x} + \mathbf{e}_i, t) - f_i(\mathbf{x}, t)\right) \frac{(\varepsilon(\mathbf{x} + \mathbf{e}_i, t) + \varepsilon(\mathbf{x}, t))}{2}. \tag{7}$$

Both equations are completely symmetric, as of course the amount of fluid leaving one cell, has to enter the other one and vice versa, which means that $\Delta m_i(\mathbf{x}) = -\Delta m_{\bar{i}}(\mathbf{x} + \mathbf{e}_i)$. For interface and neighboring fluid cells the mass change has to conform to the DFs exchanged during streaming, as for fluid cells no additional computations need to be performed. Their fluid fraction is always equal to one, and their mass equals their current density. For interface cells the mass change values for all directions are added to the current mass, resulting in the mass for the next time step:

$$m(\mathbf{x}, t + \Delta t) = m(\mathbf{x}, t) + \sum \Delta m_i(\mathbf{x}, t + \Delta t). \tag{8}$$

### 4.2. Free surface boundary conditions

As described above, the DFs of empty cells are never accessed. However, interface cells always have empty cell neighbors. Thus during the stream

| | standard cell at $(\mathbf{x}+\mathbf{e}_i)$ | no fluid neighbors at $(\mathbf{x}+\mathbf{e}_i)$ | no empty neighbors at $(\mathbf{x}+\mathbf{e}_i)$ |
|---|---|---|---|
| standard cell at $(\mathbf{x})$ | $(f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)-f_i(\mathbf{x},t))$ | $f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)$ | $-f_i(\mathbf{x},t)$ |
| no fluid neighbors at $(\mathbf{x})$ | $-f_i(\mathbf{x},t)$ | $(f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)-f_i(\mathbf{x},t))$ | $-f_i(\mathbf{x},t)$ |
| no empty neighbors at $(\mathbf{x})$ | $f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)$ | $f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)$ | $(f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)-f_i(\mathbf{x},t))$ |

**Table 1:** *Substituting the first term of Equation 7 with the appropriate one given here can force undesired interface cells to be filled or emptied.*

step only DFs from fluid cells or other interface cells can be streamed normally, while the DFs that would come out of the empty cells need to be reconstructed from the boundary conditions at the free surface. These can be handled on a per cell basis in the LBM, and do not require e.g. additional ghost layers around the interface. It is assumed, that the atmosphere has a pressure of $\rho_A = 1$, and that the fluid has a much lower kinematic viscosity. So the gas at at the interface simply flows in the direction it is pushed by the fluid. In terms of distribution functions, this means that if at $(\mathbf{x}+\mathbf{e}_i)$ there is an empty cell:

$$f_{\bar{i}}'(\mathbf{x},t+\Delta t) = f_i^{eq}(\rho_A,\mathbf{u}) + f_{\bar{i}}^{eq}(\rho_A,\mathbf{u}) - f_i(\mathbf{x},t), \tag{9}$$

where $\mathbf{u}$ is the velocity of the cell at position $(\mathbf{x})$ and time $t$ according to Equation 2. The pressure of the atmosphere onto the fluid interface is introduced by using $\rho_A$ for the density of the equilibrium DFs. Applying Equation 9 to all directions with empty neighbor cells would result in a full set of DFs for interface cells. However, to balance the forces on each side of the interface, the DFs coming from the direction of the interface normal are also reconstructed. Thus if the DF $f_i$ would be streamed from an empty cell, or if

$$\mathbf{n}\cdot\mathbf{e}_{\bar{i}} > 0 \quad \text{with} \quad \mathbf{n} = \frac{1}{2}\begin{pmatrix} \varepsilon(\mathbf{x}_{\text{j-1,k,l}}) - \varepsilon(\mathbf{x}_{\text{j+1,k,l}}) \\ \varepsilon(\mathbf{x}_{\text{j,k-1,l}}) - \varepsilon(\mathbf{x}_{\text{j,k+1,l}}) \\ \varepsilon(\mathbf{x}_{\text{j,k,l-1}}) - \varepsilon(\mathbf{x}_{\text{j,k,l+1}}) \end{pmatrix} \tag{10}$$

holds, $f_i$ is reconstructed using Equation 9. Here $\mathbf{x}_{\text{j,k,l}}$ simply denotes the position of the cell at plane $l$, row $k$ and column $j$ in the array. So the normal is approximated with central differences of the fluid fraction in each spatial direction.

Now all DFs for the interface cell are valid, and the standard collision is performed (Equation 4). The density, that was calculated during collision, is now used to check whether the interface cell filled or emptied during this time step:

$$\begin{aligned} m(\mathbf{x},t+\Delta t) &> (1+\kappa)\rho(\mathbf{x},t+\Delta t) \;\rightarrow\; \text{cell filled}, \\ m(\mathbf{x},t+\Delta t) &< (0-\kappa)\rho(\mathbf{x},t+\Delta t) \;\rightarrow\; \text{cell emptied}. \end{aligned} \tag{11}$$

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

### 4.3. Flag re-initialization

This step takes place when all cells have been updated, and has to take care of two important things: the layer of interface cells has to be closed again once the filled and emptied interface cells have been converted into their respective types, and the conservation of mass has to be maintained during the conversion. As empty cells always have a mass of 0 while fluid cells have a mass equal to their density, interface cells that have filled or emptied according to Equation 11 usually have an excess mass upon conversion, that needs to be distributed to the neighboring interface cells.

First the neighborhood of all filled cells is prepared. All neighboring empty cells are converted to interface cells. For each of these the average density $\rho^{avg}$ and velocity $\mathbf{v}^{avg}$ of the surrounding fluid and interface cells is computed, and the DFs of the empty cells are initialized with the equilibrium $f_i^{eq}(\rho^{avg},\mathbf{v}^{avg})$. Here it is necessary to remove any interface cells that are needed as boundary for a filled cell from the list of emptied interface cells. During the same pass, the flag of the filled cells is changed to fluid. Likewise, for all emptied cells the surrounding fluid cells are converted to interface cells, simply taking the former fluid cell's DFs for each corresponding new interface cell. Furthermore, the emptied interface cells are now marked as being empty. In a second pass, the excess mass $m^{ex}$ is distributed among the surrounding interface cells for each emptied and filled cell. $m^{ex}$ is equal to the mass of the cell $m$ for emptied cells (according to Equation 11 this value is negative), and can be calculated as $(m-\rho)$ for filled cells.

Negative mass values in emptied interface cells, like the mass values larger than the density in filled ones, mean that the fluid interface moved beyond the current cell during the last time step. To account for this, the mass is not distributed evenly among the surrounding interface cells, but weighted according to the direction of the interface normal $\mathbf{n}$ (which is computed as in Equation 10):

$$m(\mathbf{x}+\mathbf{e}_i) \;=\; m(\mathbf{x}+\mathbf{e}_i) + m^{ex}(\nu_i/\nu_{total}). \tag{12}$$

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

$$\begin{aligned} \nu_i &= \begin{cases} \mathbf{n}\cdot\mathbf{e}_i & \text{if } \mathbf{n}\cdot\mathbf{e}_i > 0 \\ 0 & \text{otherwise} \end{cases} && \text{for filled cells, and} \\ \nu_i &= \begin{cases} -\mathbf{n}\cdot\mathbf{e}_i & \text{if } \mathbf{n}\cdot\mathbf{e}_i < 0 \\ 0 & \text{otherwise} \end{cases} && \text{for emptied cells.} \end{aligned} \tag{13}$$

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

### 4.4. Interface cell artifacts

The algorithm described so far is already usable to animate free surfaces. However, it can happen that single interface cells are left behind when the fluid moves on, or that interface cells get enclosed in fluid. Although these cases do not perturb the fluid simulation, they are visible as artifacts. To effectively get rid of these problems the following rules can be added to the algorithm. The basic idea is to force leftover interface cells without fluid neighbors to empty, and force interface cells without empty neighbors to fill. This can be done by substituting the term $(f_{\bar{i}}(\mathbf{x}+\mathbf{e}_i,t)-f_i(\mathbf{x},t))$ from Equation 7 according to Table 4. In rare cases, where these cells still remain interface cells, they can simply be treated as filled or emptied cells. Thus, if a cell has no fluid neighbors and its mass drops below $(0.1\cdot\rho)$, it is treated as having emptied in this

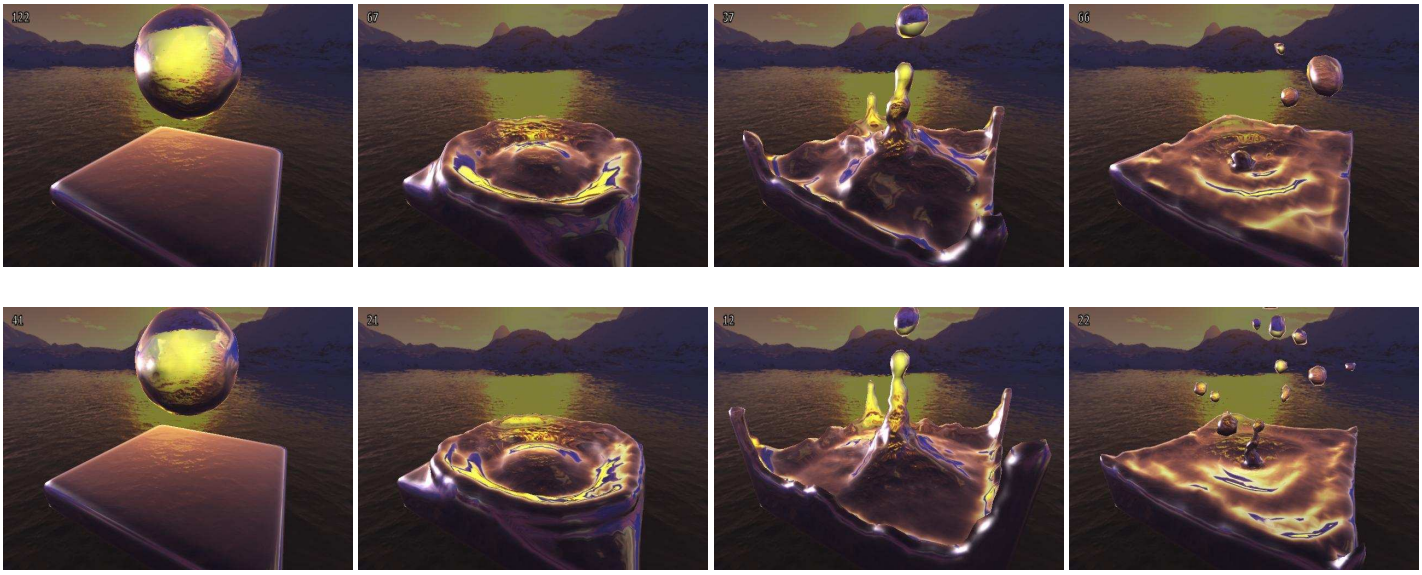

**Figure 3:** *Two animations of a single drop falling into a pool of fluid. The upper row of pictures uses a domain size of* $28^3$ *running with an average of 76 frames per second, while the lower one uses a resolution of* $38^3$ *with an average frame rate of 27.*

time step. Likewise cells with no fluid neighbors and a mass larger than $(0.9 \cdot \rho)$ are treated as full.

## 5. Adaptive time steps

As the maximum velocities for an animation are often not a priori known, the parameterization of the LBM can be difficult. Very small step sizes are often required to keep the method stable. This section describes how to alleviate this problem by allowing a resizing of the time step size, e.g. if the velocities in the simulation become too large. Validation of this method can be found in [TPR*05], while here we will focus on detail necessary for implementation. None of the equations so far contains the size of the time step, except for distinguishing the values of the two arrays. Thus, the DFs have to be explicitly recalculated to account for a new time step size.

Given an initial simulation setup with a value for $\omega$ 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:

$$|\mathbf{u}_{max}| > \frac{1}{6}/\xi \ , \text{ with } \ \xi = \frac{4}{5}. \tag{14}$$

We have chosen $1/6$ as the velocity threshold, as it is the half of $1/3$, at which point the DFs according to Equation 3 would become negative. If Equation 14 holds, we choose a new step size of

$$\Delta t_n = \xi \Delta t_o, \tag{15}$$

where $\Delta t_o$, the old step size is initially equal to 1. 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 time step size. As for LBM the value of $\omega$ also depends on the size of the time step, it changes according to:

$$\omega_n = 1/\left[ s_t \left( \frac{1}{\omega_o} - \frac{1}{2} \right) + \frac{1}{2} \right], \tag{16}$$

with $s_t = \Delta t_n / \Delta t_o$. The new acceleration in each time step is then

$$\mathbf{g}_n = s_t^2 \mathbf{g}_o. \tag{17}$$

To account for the new time step size, the velocity and also the density deviation from the median density $\rho_{med}$ have to be rescaled for each cell. Hence, after calculating $\rho_o$ and $\mathbf{u}_o$ as usual with Equation 2 for an interface or fluid cell, the new values can be computed with:

$$\rho_n = s_t \ (\rho_o - \rho_{med}) + \rho_{med} \ \ , \ \ \mathbf{u}_n = s_t \ \mathbf{u}_o. \tag{18}$$

For interface cells, this also changes the values of $m$ and $\varepsilon$:

$$m_n = m_o(\rho_o/\rho_n) \ \ , \ \ \varepsilon_n = m_n/\rho_n \tag{19}$$

The median density can be calculated from the total fluid volume $V$ and the total mass $M$ as $\rho_{med} = V/M$. The total volume is calculated by summing the values of $\varepsilon$ over all cells (for fluid cells $\varepsilon = 1$), while the $M$ is the sum of all masses (equal to the cell density for fluid cells).

As each DF contains information about the current deviation of the particles in the cell from the equilibrium, these non-equilibrium parts have to be rescaled as well. Using $\rho_n$ and $\mathbf{v}_n$ the final DFs $f_i^\star$ of the current cell can be calculated with:

$$\begin{aligned} f_i^\star &= s_f \ \left[ f_i^{eq}(\rho_o, \mathbf{u}_o) + s_\omega \ \left( f_i - f_i^{eq}(\rho_o, \mathbf{u}_o) \right) \right] \ \text{, where} \\ s_f &= f_i^{eq}(\rho_n, \mathbf{u}_n) \ / \ f_i^{eq}(\rho_o, \mathbf{u}_o) \\ s_\omega &= s_t(\omega_o \ / \ \omega_n). \end{aligned} \tag{20}$$

The values $f_i^\star$ are now stored together with $m_n$ and $\varepsilon_n$, and the next cell in the grid is treated. Of course, the time step can also be enlarged again once the fluid comes to rest. If

$$|\mathbf{u}_{max}| < \xi \frac{1}{6} \tag{21}$$

we set the step size to $\Delta t_n = \Delta t_o / \xi$. Note that a smaller time step results in a higher value of $\omega$. This means that there is a lower limit for the size of a time step – for stability we require $\omega < 1.99$. On the other hand, the time step size is limited by acceleration due to the external force per time step. For the examples shown in Section 6 we have chosen $|\mathbf{g}| < 0.001$. Note that it might be necessary to reduce this value if more fluid is added, as for the test case shown in Figure 4 . The easiest way to perform the resizing of the time step, is to track the maximum velocity during an update, and in a separate loop re-scale the DFs if a time step change is

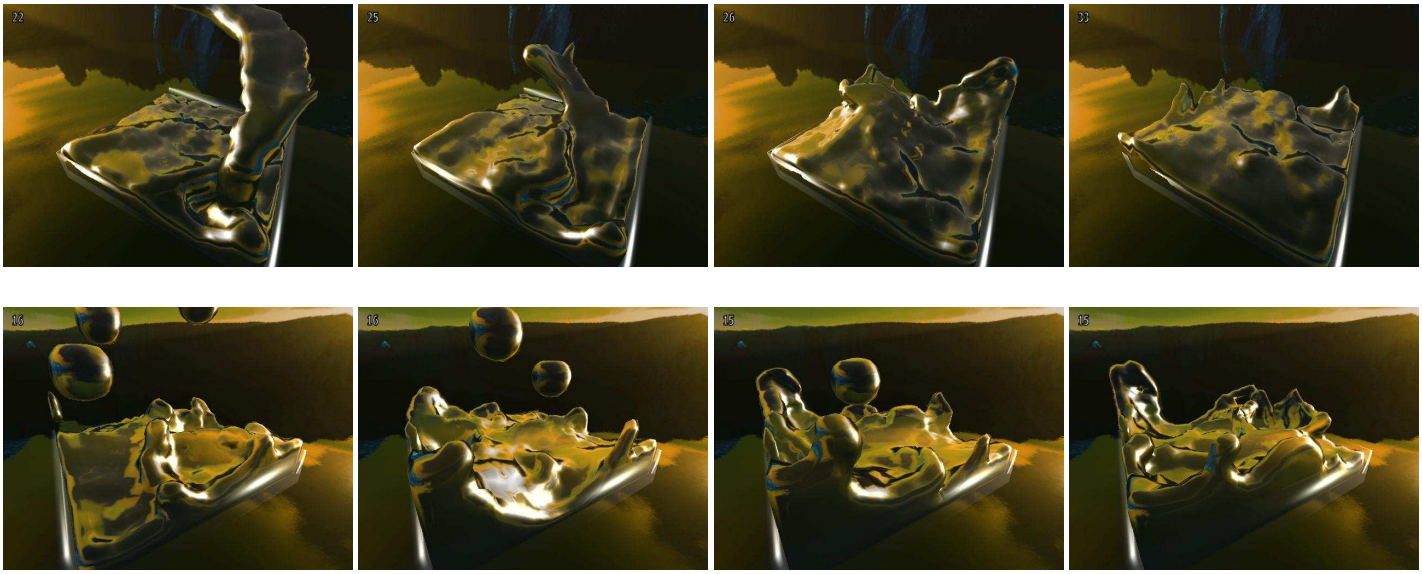

**Figure 4:** *A stream of fluid and two drops, in the top and bottom row, respectively, hit a rectangular container partly filled with fluid. Both are screenshots from our sample application, that allows the user to interactively place the drops.*

at hand. The rescaling could also be combined with the next collision of each cell, however, as the rescaling requires only 1% of the overall computational time, this would not result in a noticeable speedup.

The DF rescaling procedure described here is similar to that of LBM with grid refinement, details can be found in e.g. [FH98]. In addition to the enhanced stability due to the adaptive time steps, the animations shown in Figure 3 would run roughly 1.5 and 2.3 times slower, for the small and the large test case, respectively, without the adaptive resizing of the time step.

## 6. Results

The capabilities of the algorithm will be shown using four different setups, two with obstacles, and two without. All of them use values for $\omega$ between 1.85 and 1.95. The pictures are screenshots from real-time calculations performed on a standard Pentium 4 CPU (Northwood core) with 3.0 GHz, 512KB Level 2 Cache, and a state of the art graphics card. The latter, however, was not a limiting factor for the shown test cases. The number of frames per second can be seen in the upper left corner of each picture. It includes the calculation of the fluid movement and the visualization of the surface using a marching cubes algorithm. We simply use the fluid fraction values $\varepsilon$ and triangulate the isosurface at $\varepsilon = 0.5$. As the values of $\varepsilon$ are cut off at 0 or 1 for empty or fluid cells, respectively, we perform a filtering step before triangulation to acquire more accurate normals and a smoother appearance of the fluid surface. Including the triangulation the whole visualization requires ca. 10% of the total computational work for the shown animations. For most of them the outer rectangular walls of the domain are invisible to more clearly show the motion of the fluid inside.

The screenshots of Figure 3 are from a test case where a single drop is falling into the center of a pool of liquid. For the bottom row of pictures the size of the computational domain is $38^3$, with on average more than 10000 interface or fluid cells. This number is also given for all animations of the accompanying video, as it, together with the occurring maximum velocities, determines the overall performance. The animation of Figure 3 runs with an average frame rate of 27, which drops to 11 once the waves from all 4 corners of the domain splash together in the middle, resulting in high upward velocities. Calculating the same animation with a resolution of $28^3$ (top row of Figure 3) and on average more than 4000 used cells results in a similar fluid motion. In this case the minimal and average frame rate are 35 and 76, respectively. For the $38^3$ case, the simulation itself with 2500 LBM steps takes 16.4 seconds on the Pentium 4 system. Using an Athlon 64 4000 with 2.4 GHz and 1MB Level 2 Cache, the same calculations can be performed in 13.1 seconds. Depending on the current ratio of interface and fluid cells, our implementation can handle more than 2 million cell updates per second, and up to 3 million updates on the Athlon 64 system. To gain this performance, it is important to optimize the flag array tests, and unroll loops over the 19 distribution functions for standard fluid cells.

Screenshots from our interactive demo application are shown in Figure 4. Here a user can paint drops or lines of fluid into the domain with the mouse, resulting in turbulent and chaotic flow patterns. In both cases the average frame rate drops towards the end of each animation, when around 10000 cells used. However, with frame rates between 20 and 30 the application remains very responsive.

Figure 5 and 6 show animations with obstacles in the domain. In the first case, fluid is poured into one corner of a Z-shaped domain, filling it with fluid. The second example is again from an interactive program run, and shows drops of fluid falling into a bowl shaped obstacle in the middle of the domain. Especially the latter case results in very complex flows. Here the size of the domain is $44^3$ with up to 6000 cells being filled with fluid. The average frame rate is over 40 as the fluid always comes to rest between the subsequent drops. The high velocities near the end of the animation, when more than 15 drops are falling in short intervals, are successfully handled by the adaptive time stepping, and only result lower frame rates.

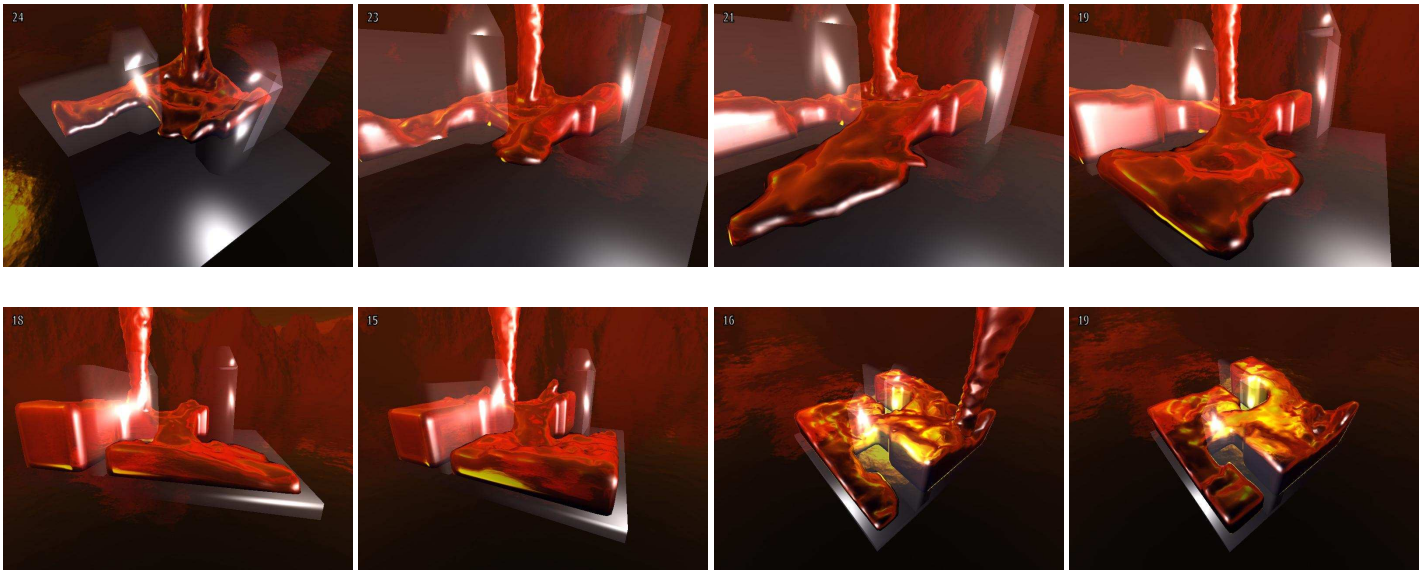

**Figure 5:** *A stream of fluid fills up a Z-shaped domain, with an average frame rate of 26. In the end more than 11000 cells of the* $36^3$ *grid are filled with fluid.*

## 7. Conclusion and Outlook

We have presented a simple, explicitly mass conserving algorithm for simulating fluids with free surfaces. Its strength are the computational efficiency, which allows simulations with reasonable domain sizes at interactive frame rates. Furthermore, the algorithm produces physically correct results, and includes the tracking of the fluid surface without the need for additional particles or markers. As it is part of the LBM framework, the algorithm is ready to be combined with previous research in this area, e.g. adaptive grid coarsening or rigid body simulation coupling.

We are currently working on efficient algorithms to include surface tension into the simulation either from the fluid fractions themselves, or from the triangulated surface. Moreover, we are trying to improve the surface reconstruction. The triangulation of the isosurface does not accurately represent the volume of the fluid, and the visual appearance of the triangulation could be improved as well.

## 8. Acknowledgements

This research is supported by the DFG Graduate College GRK-244 *3-D Image Analysis and Synthesis*. We would like to thank T. Pohl, and M. Öchsner for their work on the *FreeWIHR* project, and furthermore G. Greiner, M. Stamminger, F. Firsching, A. Borsdorf and S. Thürey for help with various parts of this research.

## References

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and Rendering of Complex Water Surfaces. *Proceedings of ACM SIGGRAPH 2002* (2002).

[FAMO99] FEDKIW R. P., ASLAM T., MERRIMAN B., OSHER S.: A non-oscillatory Eulerian approach to interfaces in multimaterial flows. *Journal of Computational Physics 152* (1999), 457 – 492.

[FdH*87] FRISCH U., D'HUMIÈRES D., HASSLACHER B., LALLEMAND P., POMEAU Y., RIVERT J.-P.: Lattice gas hydrodynamics in two and three dimensions. *Complex Systems 1* (1987), 649–707.

[FH98] FILIPPOVA O., HÄNEL D.: Grid Refinement for Lattice-BGK models. *Journal of Computational Physics 147* (1998).

[GKT*04] 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. *Computers & Fluids* (2004).

[GRZZ91] GUNSTENSEN A. K., ROTHMAN D. H., ZALESKI S., ZANETTI G.: Lattice Boltzmann model for free-surface flow and its application to filling process in casting. *Phys. Rev. A 43* (1991).

[GS03] GINZBURG I., STEINER K.: Lattice Boltzmann model for free-surface flow and its application to filling process in casting. *Journal of Computational Physics 185/1* (2003).

[HL97a] HE X., LUO L.-S.: Lattice boltzmann model for the incompressible navier-stokes equations. *Journal of Statistical Physics 88* (1997).

[HL97b] HE X., LUO L.-S.: A priori derivation of lattice boltzmann equation. *Phys. Rev. E 55* (1997).

[HN81] HIRT C. W., NICHOLS B. D.: Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *Journal of Computational Physics 39* (1981).

[KS00] KÖRNER C., SINGER R.: Processing of Metal Foams - Challenges and Opportunities. *Advanced Engineering Materials 2(4)* (2000), 159–65.

[KTS02] KÖRNER C., THIES M., SINGER R. F.: Modeling of Metal Foaming with Lattice Boltzmann Automata. *Advanced Engineering Materials* (2002).

[LL00] LALLEMAND P., LUO L.-S.: Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E 61* (2000).

[LLS00] LOCKARD D. P., LUO L.-S., SINGER B. A.: *Evaluation of the lattice-Boltzmann equation solver PowerFLOW for aerodynamic applications*. Tech. rep., ICASE, 2000.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based

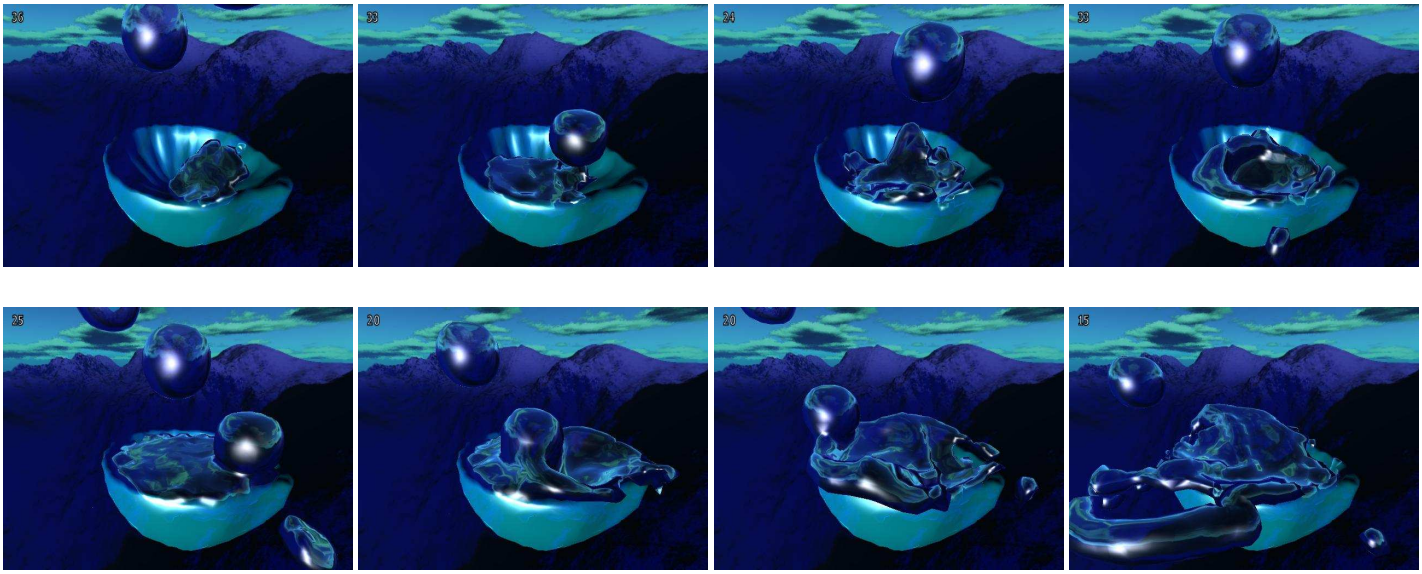

**Figure 6:** *Several interactively placed drops of fluid hit a bowl-shaped obstacle, resulting in complex splashes.*

fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation* (2003).

[PKW*03] POHL T., KOWARSCHIK M., WILKE J., IGLBERGER K., RÜDE U.: Optimization and profiling of the cache performance of parallel lattice boltzmann codes. *Parallel Processing Letters* (2003).

[PTD*04] 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).

[Sta03] STAM J.: Real-Time Fluid Dynamics for Games. *Proceedings of the Game Developer Conference* (March 2003).

[Suc01] SUCCI S.: *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001. 2 copies.

[Sus03] SUSSMAN M.: A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *Journal of Computational Physics 187/1* (2003).

[Thü03] THÜREY N.: A Lattice Boltzmann method for single-phase free surface flows in 3D. Masters thesis, 2003.

[TPR*05] THÜREY N., POHL T., RÜDE U., ÖCHSNER M., KÖRNER C.: Optimization and Stabilization of LBM Free Surface Flow Simulations using Adaptive Parameterization. *To appear in the Proceedings of ICMMES 2004* (2005).

[TR04] THÜREY N., RÜEDE U.: Free surface lattice-boltzmann fluid simulations with and without level sets. IOS Press, pp. 199–208. Workshop on Vision, Modelling, and Visualization VMV.

[WZF*03] WEI X., ZHAO Y., FAN Z., LI W., YOAKUM-STOVER S., KAUFMAN A.: Blowing in the Wind. *ACM Siggraph/Eurographics Symposium on Computer Animation* (2003).