Chapter 11
Boundary Conditions for Fluid-Structure Interaction

Abstract After reading this chapter, you will have insight into a large number of more complex lattice Boltzmann boundary conditions, including advanced bounce-back methods, ghost methods, and immersed boundary methods. These boundary conditions will allow you to simulate things like curved boundaries, flows in media with sub-grid porosity, rigid but moveable objects immersed in the fluid, and even flows with deformable objects such as red blood cells.

Boundary conditions play a paramount role in hydrodynamics. Chapter 5 concerns itself with the definition and conceptual introduction of boundary conditions, and it provides an overview of boundary conditions for relatively simple solid geometries, flow inlets and outlets and periodic systems. Here, we turn our attention to resting and moving boundaries with complex shapes (Sect. 11.1). It is nearly impossible to give an exhaustive overview of all available boundary conditions for fluid-structure interaction in the LBM. We will therefore focus on the most prominent examples: bounce-back methods in Sect. 11.2, extrapolation methods in Sect. 11.3 and immersed-boundary methods in Sect. 11.4. We provide a list of comparative benchmark studies and an overview of the strengths and weaknesses of the discussed boundary conditions in Sect. 11.5.

11.1 Motivation

Many works about boundary conditions in the LBM assume flat, resting and rigid boundaries. We have reviewed a selection of those methods in Chap. 5. But our experience tells us that only a small number of boundaries in fluid dynamics obey these assumptions. In reality, most boundaries are curved, some can move and others are deformable. Prominent examples are porous media, curved surfaces of cars and planes in aerodynamics, suspensions (e.g. clay, slurries) or deformable objects (e.g. cells, wings, compliant containers). Analytical solutions are often impossible to obtain, which makes computer simulations an indispensible tool. This challenge led to a remarkable variety of proposed methods to model complex boundaries in LB simulations.

© Springer International Publishing Switzerland 2017

T. Krüger et al., *The Lattice Boltzmann Method*, Graduate Texts in Physics,
DOI 10.1007/978-3-319-44649-3_11

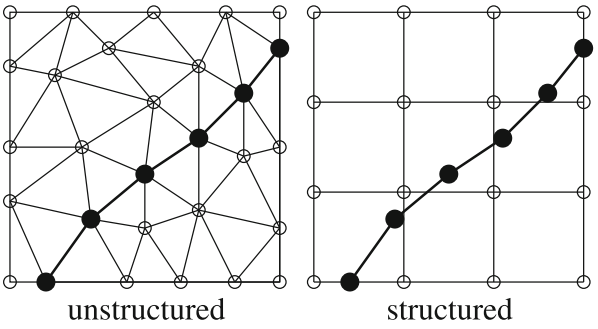

Fig. 11.1 Unstructured and structured meshes. The same boundary problem (*solid black circles connected with thick lines*) can be treated by, for example, an unstructured (*left*) or structured (*right*) approach. The former requires remeshing if the boundary moves, the latter leads to interpolations or extrapolations near the boundary

In order to accurately describe a complex domain, there are essentially two options (Fig. 11.1). The first approach is to formulate the problem in a coordinate system which fits the shape of the boundary. This leads to curvilinear or body-fitted meshes where the boundary treatment itself is trivial. However, this way we lose the advantages of the simple cartesian grid. For example, if the boundary shape changes in time, the curvilinear coordinate system also changes or remeshing becomes necessary. This can be a challenging and time-consuming task [1]. The alternative is to retain the cartesian structure of the bulk geometry, but then we have to introduce special procedures to account for the complex shape of the boundary which does generally not conform with the underlying lattice structure. In the end, this leads to interpolation and extrapolation boundary schemes.

Since most LB algorithms take advantage of the cartesian grid, the second route is usually preferred. First, it is easier to correct only the behaviour of the boundary nodes than touching all bulk nodes. Secondly, remeshing of the bulk involves interpolations in the entire numerical domain, which can lead to detrimental numerical viscosities (hyperviscosities) and a loss of exact mass/momentum conservation. More details about LB for non-cartesian geometries (i.e. curvilinear structured meshes or unstructured meshes) are provided in, e.g., [2–7]. It is therefore less harmful to use interpolations only in the vicinity of the boundaries. In this chapter we will exclusively address boundary treatments of the second kind, where the underlying lattice structure is not changed.

There are different types of problems which are typically encountered in connection with off-lattice boundaries. We can identify three main categories:

- stationary rigid obstacles (e.g. porous media, microfluidic devices, flow over stationary cylinders)
- moving rigid obstacles (e.g. suspensions of non-deformable particles, oscillating cylinder, rotating turbine blades)

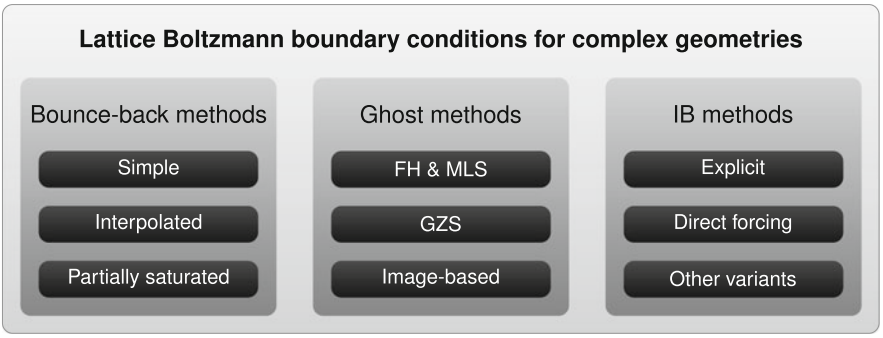

Fig. 11.2 Overview of boundary conditions for complex geometries in LB simulations as presented in this chapter. We can roughly distinguish between bounce-back, ghost and immersed-boundary (IB) methods. Each of them has several flavours. A large selection of those is covered in the following sections

- moving deformable obstacles (e.g. flexible wings, red blood cells, compliant channels)

No single numerical boundary treatment works best for all of them. It is therefore worth to properly categorise the problem first, identify the main challenge and then "shop around" and look for the most suitable boundary treatment for the problem at hand. This chapter helps the reader to understand what the differences of the available methods are, when they are applicable and what their advantages and disadvantages are.

There exists a zoo of curved boundary conditions for LB simulations. We can only cover the most popular ones in any depth, but we will provide references to a wider range of boundary conditions in passing. Figure 11.2 shows an overview of the boundary conditions discussed here. For the sake of compactness, we only consider single-phase fluids. Note that everything said in this chapter does equally apply to 2D and 3D systems.

11.2 Bounce-Back Methods

The most famous and certainly easiest boundary condition for LB simulations is bounce-back (Sect. 5.3.3). Many researchers believe that its locality, simplicity and efficiency should be retained even in the presence of complex boundary shapes. Therefore, the obvious way is to approximate a curved boundary by a staircase (Sect. 11.2.1). This can lead to some problems, in particular a reduction of the numerical accuracy. For that reason, improved and interpolated bounce-back schemes have been proposed (Sect. 11.2.2). Another variant to account for complex geometries is the partially saturated method (Sect. 11.2.3). A problem related to staircase and interpolated bounce-back BCs is the destruction and creation of fluid

sites if the boundaries move. We will discuss the creation of so-called *fresh nodes* in Sect. 11.2.4. Finally we will elaborate on the calculation of the wall shear stress in the presence of complex boundaries (Sect. 11.2.5). We recommend reading [8–15] to understand bounce-back methods in greater detail.

11.2.1 Simple Bounce-Back and Staircase Approximation

One of the motivations to simulate complex geometries is to study flows in porous media. The simplest way to introduce curved or inclined boundaries in LB simulations is through a *staircase approximation* of the boundary and the bounce-back scheme, often called *simple bounce-back* (SBB, Sect. 5.3.3). This is illustrated in Fig. 11.3. The advantages are obvious: everything lives on the lattice, and SBB is fast and easy to implement. The problem becomes more complex when the boundaries can move, which requires the destruction and creation of fluid sites (Sect. 11.2.4).

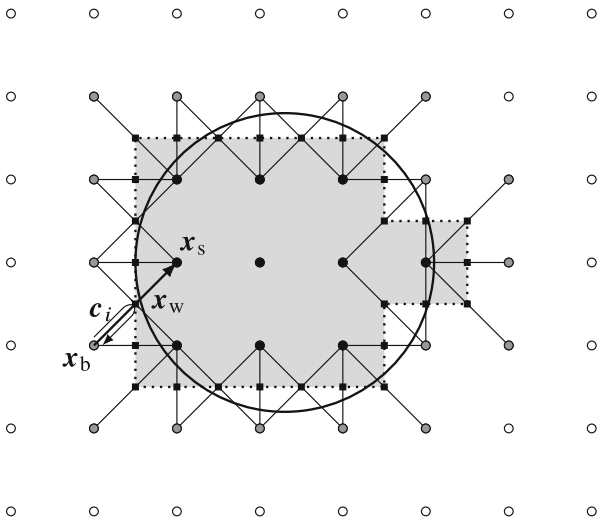

Fig. 11.3 Staircase approximation of a circle. A circle (here with an unrealistically small radius $r = 1.8\Delta x$) can be discretised on the lattice by identifying exterior fluid nodes (*white circles*), external boundary nodes (*grey circles*) and interior solid nodes (*black circles*) first. Any lattice link $\boldsymbol{c}_i$ connecting a boundary and a solid node is a cut link (*lines*) with a wall node (*solid squares*) in the middle. The resulting staircase shape is shown as a *grey-shaded area*. Populations moving along cut links $\boldsymbol{c}_i$ (defined as pointing inside the solid) from $\boldsymbol{x}_b$ to $\boldsymbol{x}_s$ are bounced back at $\boldsymbol{x}_w$

11.2.1.1 Revision of the Halfway Bounce-Back Method

In the following we will only consider the halfway bounce-back scheme: an incoming (post-collision) population $f_i^\star(\boldsymbol{x}_\mathrm{b}, t)$ which would propagate through a wall from a boundary node[1] $\boldsymbol{x}_\mathrm{b}$ to a solid node $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{c}_i\Delta t$ is instead reflected halfway to the solid node at the wall location $\boldsymbol{x}_\mathrm{w} = \boldsymbol{x}_\mathrm{b} + \frac{1}{2}\boldsymbol{c}_i\Delta t$ at time $t + \frac{1}{2}\Delta t$ and returns to $\boldsymbol{x}_\mathrm{b}$ as

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t + \Delta t) = f_i^\star(\boldsymbol{x}_\mathrm{b}, t) - 2w_i\rho\frac{\boldsymbol{u}_\mathrm{w}\cdot\boldsymbol{c}_i}{c_\mathrm{s}^2} \tag{11.1}$$

where $\boldsymbol{u}_\mathrm{w} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{w}, t + \frac{1}{2}\Delta t)$ is the velocity of the wall, ρ is the fluid density at $\boldsymbol{x}_\mathrm{w}$ and $\bar{i}$ is defined by $\boldsymbol{c}_{\bar{i}} = -\boldsymbol{c}_i$. In practical implementations, ρ is often taken as the fluid density at $\boldsymbol{x}_\mathrm{b}$ or the average fluid density instead (cf. Sect. 5.3.3).

The halfway bounce-back scheme requires detection of all lattice links $\boldsymbol{c}_i$ intersecting the boundary. If the boundary is stationary, this has to be done only once.

We can compute the momentum exchange at the wall based on the incoming and bounced back populations alone by using the momentum exchange algorithm (MEA, Sect. 5.4.3). Here we will briefly revise the MEA for the simple bounce-back method. The first step is to evaluate the incoming and bounced back populations $f_i^\star$ and $f_{\bar{i}}$ at each boundary link. The total momentum exchange between the fluid and the solid during one time step is given by (5.79):

$$\begin{aligned}\Delta\boldsymbol{P} &= \Delta x^3\sum_{\boldsymbol{x}_\mathrm{w},i}\left(f_i^\star(\boldsymbol{x}_\mathrm{w} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) + f_{\bar{i}}(\boldsymbol{x}_\mathrm{w} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t + \Delta t)\right)\boldsymbol{c}_i \\ &= \Delta x^3\sum_{\boldsymbol{x}_\mathrm{w},i}\left(2f_i^\star(\boldsymbol{x}_\mathrm{w} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) - 2w_i\rho\frac{\boldsymbol{u}_\mathrm{w}\cdot\boldsymbol{c}_i}{c_\mathrm{s}^2}\right)\boldsymbol{c}_i\end{aligned} \tag{11.2}$$

where the sum runs over all incoming links $\boldsymbol{c}_i$ (pointing from the fluid into the solid) intersecting the wall at $\boldsymbol{x}_\mathrm{w}$. Accordingly, the total angular momentum exchange during one time step is

$$\Delta\boldsymbol{L} = \Delta x^3\sum_{\boldsymbol{x}_\mathrm{w},i}\left(2f_i^\star(\boldsymbol{x}_\mathrm{w} - \tfrac{1}{2}\boldsymbol{c}_i\Delta t, t) - 2w_i\rho\frac{\boldsymbol{u}_\mathrm{w}\cdot\boldsymbol{c}_i}{c_\mathrm{s}^2}\right)(\boldsymbol{x}_\mathrm{w} - \boldsymbol{R})\times\boldsymbol{c}_i. \tag{11.3}$$

$\boldsymbol{R}$ is a reference point. If the torque acting on a particle is computed, the reference point is the particle's centre of mass.

[1] We use the same notation as in Chap. 5: solid nodes are inside the obstacle, boundary nodes are outside the obstacle but have at least one solid neighbour, and fluid nodes are those without a solid neighbour (see Fig. 11.3).

The MEA works for any geometry approximated by the simple bounce-back scheme, including the staircase shown in Fig. 11.3. The tedious part is the identification of all cut links pointing from a boundary to a solid node and obtaining the wall velocity $\boldsymbol{u}_{\mathrm{w}}$ at each of the wall locations $\boldsymbol{x}_{\mathrm{w}}$.

11.2.1.2 Stationary Boundaries

There is a large number of applications where the flow in a complex stationary geometry has to be simulated. Examples are flows in porous media or blood flow in the vascular system (Fig. 11.4). Those geometries can by obtained by, for example, CT or MRI scans. Due to the large surface and complex shape of those geometries, it is preferable to use a simple and fast boundary condition algorithm, such as SBB.

Back in the 90s, Ginzburg and Adler [9] presented a very careful analysis of halfway SBB with several important conclusions. A more updated discussion of this work can be found in [15]. Apart from developing general theoretical tools to study boundary conditions, one contribution was the understanding of the numerical mechanism leading to the velocity slip at the wall. The exact location where the no-slip condition is satisfied is not a pre-determined feature; it rather depends on the specific choice of the relaxation rate(s). Furthermore, the above-mentioned defect is anisotropic with respect to the underlying lattice structure, i.e. the slip velocity depends on the way the boundary is inclined. This is confirmed for a Poiseuille flow in an horizontal channel where the wall can only be *exactly* located midway between lattice nodes if $\tau/\Delta t = 1/2 + \sqrt{3/16} \approx 0.933$ (cf. Sect. 5.3.3). Contrarily, for a diagonal channel, $\tau/\Delta t = 1/2 + \sqrt{3/8} \approx 1.11$ has to be chosen. With

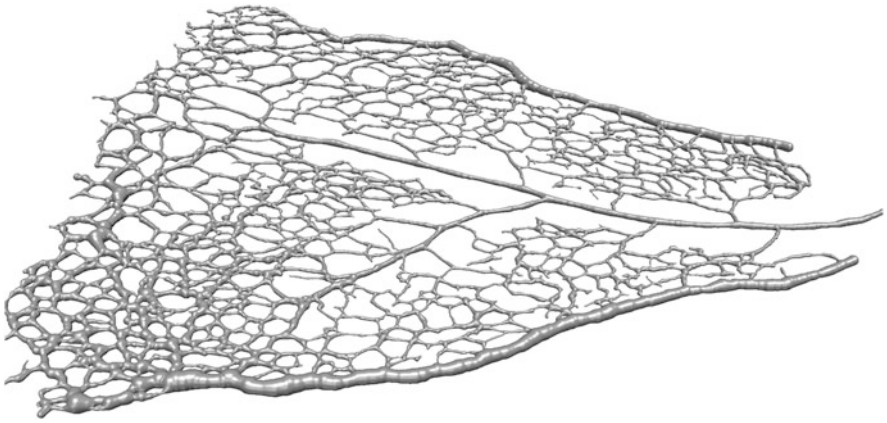

Fig. 11.4 Visualisation of a segment of the blood vessel network in a murine retina which has been used for LB simulations. This example shows the complexity of the involved boundaries. Original confocal microscope images courtesy of Claudio A. Franco and Holger Gerhardt. Luminal surface reconstruction courtesy of Miguel O. Bernabeu and Martin L. Jones. For more details see [16]

the LBGK method this τ-dependence leads to a viscosity-dependent slip velocity artefact, which is highly undesired from the physical viewpoint.

While in complex geometries the wall intersects the lattice at different positions, the SBB enforces the no-slip condition to be fixed at all lattice links. This leads to the staircase representation of the boundary, where the exact location of the no-slip surface will further depend on the choice of the relaxation time τ. In other words, the effective shape and location of the "numerical" wall will not agree with the expected boundary.

The aforementioned problems are particularly harmful in narrow domains where the distance between walls can be of the order of a few lattice units, for example in porous media flows or solid particles in suspensions. Hereby, using TRT/MRT collision operators with a properly tuned set of relaxation rates [12, 15, 17, 18], rather than BGK, helps controlling this situation. When τ is chosen close to 1 and the fluid domain is sufficiently resolved, SBB with BGK leads to acceptable results, though [19].

We also have to note that biological geometries obtained from CT or MRI scans are usually not very accurate in the first place. The resolution of those imaging techniques can be of the same order as the pore or channel size so that it may be nonsensical trying to increase the resolution of the numerical domain or choose more accurate boundary conditions. This means that SBB, although generally not the most recommendable solution, is still a good choice given the large geometrical modelling error in many applications. Furthermore, it is worth mentioning that SBB is exactly mass-conserving when used for stationary geometries of any shape; an advantage only a few higher-order accurate boundary conditions can claim (the reasons are explained in [20]). Therefore, before setting up a simulation, one should always ask whether the boundary condition is really the limiting factor in terms of accuracy.

Using **simple bounce-back (SBB)** for complex geometries generally leads to two sources of error:

1. geometrical discretisation error (modelling error) by approximating a complex shape by a staircase,
2. artificial and anisotropic slip caused by the choice of the relaxation rate(s), leading to a viscosity-dependent effect when is BGK used.

The advantages of SBB (mass conservation, ease of implementation, locality) explain why it is still a popular method.

11.2.1.3 Rigid Moving Particles

So far we have only addressed stationary boundaries with SBB. In the 90s researchers became interested in the simulation of suspensions *via* LBM. This requires the treatment of multiple rigid particles with translational and rotational degrees of freedom. One of the problems of earlier computational suspension models was the numerical cost which scaled with the square or cube of the particle number [10]. Ladd [10, 21, 22] introduced an LB-based model for suspensions of rigid particles with hydrodynamic interactions whose numerical cost scales linearly with the particle number.

Particle suspensions give rise to a plethora of physical effects and phenomena. In this section we will only focus on the algorithmic details. For physical results we refer to review articles about LB-based suspension simulations [23, 24] and the references therein.

For the sake of brevity we will not discuss lubrication forces which become necessary at high particle volume fractions. There exist several articles dealing with lubrication corrections in LB simulations [24–26]. The review by Ladd and Verberg [23], which we generally recommend to read, also descibes the use of thermal fluctuations for the simulation of Brownian motion in suspensions. Aidun and Clausen [24] have published a review about LBM for complex flows, which is an excellent starting point to learn about more recent developments.

In the following we will outline Ladd's [10, 22] idea of how to use SBB for suspensions. See also [25] for a compact and [23] for an extensive review of Ladd's method. Note that the particles in Ladd's algorithm are filled with fluid in order to avoid destruction and recreation of fluid nodes when the particles move on the lattice. The dynamics of the interior fluid is therefore fully captured. One can imagine this like a can filled with liquid concrete in an exterior fluid rather than the same can with set (and therefore solid) concrete. This is different compared to Aidun's model [27] which we will briefly describe at the end of this section.

The first step is to start with a distribution of suspended spherical particles on the lattice. For each particle, it is straightforward to work out which lattice nodes are located inside and outside of a particle (cf. Fig. 11.3). There is no conceptual difficulty in extending the model to non-spherical particles; but it will generally be more demanding to identify interior and exterior lattice nodes.

The next step is to identify all lattice links between boundary and solid nodes, i.e. those links cut by any particle surface. For moving boundaries, the list of those links has to be updated whenever the boundary configuration on the lattice changes. Generally one has to update the list every time step before propagation is performed. In the following, let $\boldsymbol{x}_\mathrm{b}$ be the location of a boundary node and $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{c}_i \Delta t$ the location of a solid node just inside the particle. The boundary link is then located at $\boldsymbol{x}_\mathrm{w} = \boldsymbol{x}_\mathrm{b} + \frac{1}{2}\boldsymbol{c}_i \Delta t$ (cf. Fig. 11.3).

Now we have to compute the wall velocity $\boldsymbol{u}_\mathrm{w}$ at each link $\boldsymbol{x}_\mathrm{w}$. From the known linear velocity $\boldsymbol{U}$ and angular velocity $\boldsymbol{\Omega}$ of the particle we can obtain

$$\boldsymbol{u}_\mathrm{w} = \boldsymbol{U} + \boldsymbol{\Omega} \times (\boldsymbol{x}_\mathrm{w} - \boldsymbol{R}) \tag{11.4}$$

where $\boldsymbol{R}$ is the particle's centre of mass.

With the known wall velocity at each link, we can compute the momentum exchange and therefore the value of all bounced-back populations. We have to take into account that a particle in Ladd's method is filled with fluid, as explained earlier, and all interior nodes participate in collision and propagation as well. This means that there are also populations streaming from the interior nodes at $\boldsymbol{x}_\mathrm{s}$ to exterior nodes at $\boldsymbol{x}_\mathrm{b}$. These populations have to be bounced back at $\boldsymbol{x}_\mathrm{w}$, too. While the populations streaming from the outside to the particle's interior are described by (11.1), we now also have to consider those populations streaming from the inside towards the exterior:

$$f_i(\boldsymbol{x}_\mathrm{s}, t + \Delta t) = f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{s}, t) - 2w_{\bar{i}}\rho\frac{\boldsymbol{u}_\mathrm{w} \cdot \boldsymbol{c}_{\bar{i}}}{c_\mathrm{s}^2} = f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{s}, t) + 2w_i\rho\frac{\boldsymbol{u}_\mathrm{w} \cdot \boldsymbol{c}_i}{c_\mathrm{s}^2}. \tag{11.5}$$

Equation (11.1) and (11.5) express that the two populations hitting a boundary link from both sides exchange a certain amount of momentum, $2w_i\rho\boldsymbol{u}_\mathrm{w} \cdot \boldsymbol{c}_i/c_\mathrm{s}^2$. This operation is obviously mass-conserving since f_i gains exactly the loss of $f_{\bar{i}}$ (or the other way around) so that the sum of both populations moving along the same link in different directions is not changed by the interaction with the boundary, at least as long the same density ρ is used in both equations. Ladd uses the average fluid density, and not the local density, for ρ.

Effectively, we can view the momentum transferred from the exterior to the interior fluid as the momentum transferred from the exterior fluid to the particle. In order to obey the global momentum and angular momentum conservation, we therefore have to update the particle momentum and angular momentum by summing up all transferred contributions. Equation (11.2) and (11.3) provide the total momentum $\Delta\boldsymbol{P}$ and angular momentum $\Delta\boldsymbol{L}$ transferred during one time step, but we have to take into account that each link has to be counted twice: once for all populations coming from the outside and once for all populations coming from the inside. This is necessary because the interior fluid participates in collision and propagation and therefore the momentum exchange.

The simplest way to update the particle properties is the forward Euler method, but more accurate and more stable methods are available, e.g. implicit time integration [23]. At each time step, the velocity and angular velocity are updated according to

$$\boldsymbol{U}(t + \Delta t) = \boldsymbol{U}(t) + \frac{\Delta\boldsymbol{P}}{M}, \quad \boldsymbol{\Omega}(t + \Delta t) = \boldsymbol{\Omega}(t) + \boldsymbol{I}^{-1} \cdot \Delta\boldsymbol{L} \tag{11.6}$$

where M and $\boldsymbol{I}$ are the particle's mass and tensor of inertia. The centre of mass is then moved according to the old or new velocity.[2] If the particles are spherical, their orientation does not have to be updated. For non-spherical particles, however, the situation is different, and several authors have suggested algorithms for this case. Aidun et al. [27], for example, use a fourth-order Runge-Kutta integration to update the particle orientation. Qi [28] employed quaternions to capture the particle orientation and a leap-frog time integration. It is noteworthy that Ladd [10] does not follow the simple scheme in (11.6). He instead averages the momentum and angular momentum transfer over two time steps before updating the particle properties. The reason for this is to reduce the undesired effect of so-called *staggered momenta* which are an artefact of lattice-based methods. We refer to [10] for a more thorough discussion of this issue (see also Sect. 5.3.3).

Ladd's algorithm [10] can be summarised in the following way:

1. Find the particle discretisation on the lattice (Fig. 11.3).
2. Identify all boundary links and compute $\boldsymbol{u}_{\mathrm{w}}$ by applying (11.4).
3. Perform collision on *all* nodes since particles are filled with fluid.
4. Propagate the populations. If a population moves along a boundary link, bounce-back this population *via* (11.1) or (11.5).
5. Compute the total momentum and angular momentum exchange according to (11.2) and (11.3).
6. Update the particle configuration, for example *via* (11.6).
7. Go back to step 1 for the next time step. There is no need to treat nodes crossing a boundary in a special way.

It is interesting to note why Ladd has chosen a link-based (halfway) rather than a node-based (fullway) bounce-back method. The simple explanation is that the link-based bounce-back leads to a "somewhat higher resolution" [10] for the same discretisation since there are more cut links than solid nodes near the particle surface. This can be easily seen in Fig. 11.3.

Ladd [10] pointed out that his method has a few disadvantages. First, the dynamics of the fluid inside the particles can affect the particle dynamics at higher Reynolds numbers where the interior fluid cannot any more be approximated by an effectively rigid medium. Furthermore, Ladd's method is limited to situations where the particle density is larger than the fluid density. Aidun et al. [27] proposed an alternative method with one major distinctive feature: the absence of fluid inside the particles. Therefore, in Aidun's approach, only the exterior fluid contributes to the momentum and angular momentum exchange in (11.2) and (11.3). Removing the fluid from the interior solves both disadvantages of Ladd's method, but it also introduces a new complexity: what happens when lattice nodes change their identity (fluid nodes become solid nodes and the other way around) when the particles move? We will come back to this point in Sect. 11.2.4. In contrast to Ladd's approach,

[2] The velocities are usually small so that the exact form of the position update is not very important.

Aidun's method does not obey global mass conservation [27]. Yin et al. [29] provide a detailed comparative study on the performance of both models.

We emphasise that several researchers have further improved the methods presented above. For example, Lorenz et al. [30], Clausen and Aidun [31] and Wen et al. [32] proposed modified versions of the momentum exchange to improve Galilean invariance.

We have only discussed *link-based* BB schemes in this section. It is possible to implement node-based BB schemes for complex geometries where the boundary velocity is enforced directly on lattice nodes, though. Behrend [33] and Gallivan et al. [34] provide discussions of the node-based BB approach. In Sect. 11.2.3 we will present partially saturated methods which are also built on the node-based BB scheme.

As pointed out by Han and Cundall [35], the **simple bounce-back (SBB) applied to moving boundaries** has its limitations compared to higher-order schemes, such as the partially saturated method (Sect. 11.2.3). This becomes most obvious when the particles are rather small (a few Δx in diameter) and move on the lattice. Eventually, the user has to decide whether the focus lies on the ease of implementation or level of accuracy. In the former case, SBB can be recommended. In the latter, a smoother boundary condition should be implemented.

11.2.2 Interpolated Bounce-Back

We will now cover interpolated bounce-back (IBB) methods which are suitable to describe curved and inclined boundaries with a higher accuracy than SBB. We emphasise the conceptual difference between IBB schemes and extrapolation-based methods (Sect. 11.3). While the idea of the former is to interpolate populations in the fluid region to perform bounce-back at a curved wall, the motivation for the latter is to create a virtual (ghost) fluid node inside the solid to compute the populations streaming out of the wall. We generally recommend reading [12, 17, 18, 36] for thorough reviews of IBB methods.

11.2.2.1 Basic Algorithm

In 2001, Bouzidi et al. [11] proposed the IBB approach for curved boundaries. The IBB is *second-order* accurate for arbitrary boundary shapes and therefore reduces the modelling error of the staircase bounce-back method which is only first-order accurate for non-planar boundaries.

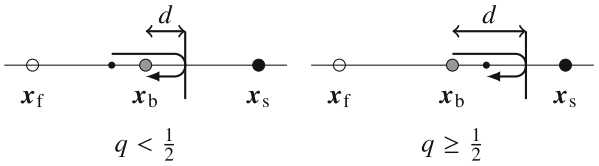

Fig. 11.5 Interpolated bounce-back cases. For $q < \frac{1}{2}$ (*left*), the distance d between the wall (*vertical line*) and the boundary node at $\boldsymbol{x}_\mathrm{b}$ is smaller than half a lattice spacing. Interpolations are required to construct the post-collision population (*small black circle*). For $q \geq \frac{1}{2}$ (*right*), d is larger than half a lattice spacing and the endpoint of the streaming population (*small black circle*) lies between the wall and $\boldsymbol{x}_\mathrm{b}$. $\boldsymbol{x}_\mathrm{s}$ denotes a solid node behind the wall, and $\boldsymbol{x}_\mathrm{f}$ is a fluid node required for the interpolation

The basic idea of IBB is to include additional information about the wall location during the bounce-back process. A boundary link $\boldsymbol{c}_i$ generally intersects the wall at a distance d between 0 and $|\boldsymbol{c}_i|\Delta t$ measured from the boundary node (Fig. 11.5). We define $q = d/(|\boldsymbol{c}_i|\Delta t) \in [0, 1)$ as the reduced wall location and note that $q = \frac{1}{2}$ holds for simple bounce-back. In the following, we introduce three lattice nodes with locations $\boldsymbol{x}_\mathrm{b}$, $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{c}_i\Delta t$ and $\boldsymbol{x}_\mathrm{f} = \boldsymbol{x}_\mathrm{b} - \boldsymbol{c}_i\Delta t$ as shown in Fig. 11.5. $\boldsymbol{x}_\mathrm{b}$ and $\boldsymbol{x}_\mathrm{s}$ are neighbouring boundary and solid nodes which are located on either side of the wall, and $\boldsymbol{x}_\mathrm{f}$ is the nearest fluid node beyond $\boldsymbol{x}_\mathrm{b}$.

The starting point of IBB is to assume that any population f_i moves a distance $|\boldsymbol{c}_i|\Delta t$ during propagation. If the population hits a wall which is modelled by the halfway bounce-back, f_i first travels a distance $|\boldsymbol{c}_i|\Delta t/2$ from the original boundary node to the wall and then another distance $|\boldsymbol{c}_i|\Delta t/2$ back to the boundary node after bounce-back. If the wall is not located halfway between lattice nodes, $q \neq \frac{1}{2}$, f_i cannot reach another lattice node. Therefore, the origin of the population is chosen such that f_i exactly reaches a lattice node. This requires interpolation to find the post-collision value of f_i as illustrated in Fig. 11.5.

Bouzidi et al. [11] proposed a linear interpolation to construct the *a priori* unknown bounced back population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t + \Delta t)$ from the post-collision values of the known populations at $\boldsymbol{x}_\mathrm{b}$ and $\boldsymbol{x}_\mathrm{f}$. The algorithm for any cut link at a resting wall reads

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t + \Delta t) = \begin{cases} 2q f_i^\star(\boldsymbol{x}_\mathrm{b}, t) + (1 - 2q) f_i^\star(\boldsymbol{x}_\mathrm{f}, t) & q \leq \frac{1}{2} \\ \frac{1}{2q} f_i^\star(\boldsymbol{x}_\mathrm{b}, t) + \frac{2q-1}{2q} f_{\bar{i}}^\star(\boldsymbol{x}_\mathrm{b}, t) & q \geq \frac{1}{2} \end{cases} . \tag{11.7}$$

Exercise 11.1 Show that both cases in (11.7) reduce to simple bounce-back for $q = \frac{1}{2}$.

There are several remarks:

- The reason for having different expressions for $q < \frac{1}{2}$ and $q \geq \frac{1}{2}$ is to ensure that $f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t + \Delta t)$ is always non-negative (given that the post-collision populations

on the right-hand-side of (11.7) are positive), which improves the stability of the algorithm.

- Bouzidi et al. [11] have also proposed a quadratic interpolation which improves results, but the method remains *second-order* accurate. This extension requires a second fluid node $\boldsymbol{x}_{\mathrm{ff}} = \boldsymbol{x}_{\mathrm{b}} - 2\boldsymbol{c}_i\Delta t$.
- The boundary slip for linear and quadratic interpolations still depends on the collision relaxation rate(s). However, unlike with the SBB, the adoption of TRT/MRT collision operators does not allow absolute control of this error; the IBB thereby always presents some degree of viscosity dependence. For strategies to render IBB *exactly* viscosity-independent we refer to [15, 37, 38].
- The IBB algorithm, like simple bounce-back, is completely decoupled from the collision step and can therefore be combined with any collision operator.
- Due to its non-local implementation, the IBB may lead to problems in very narrow gaps (e.g. in porous media simulations) where there are not enough fluid nodes between neighbouring walls to apply (11.7) (two nodes required) or the quadratic interpolation (three nodes required). The number of required nodes may be reduced by one with a judicious choice of pre- and post-collision populations within the IBB algorithm, cf. [15, 17]. For example, the IBB with linear interpolations can be written in local form. Chun and Ladd [18] have addressed this issue and proposed an alternative local boundary scheme which works in general situations and corrects some defects of the IBB, e.g. its viscosity dependence.
- Due to its interpolations, IBB is generally not mass-conserving. There exist approaches to (partially) remedy this shortcoming, e.g. [36].

The interpolated bounce-back algorithm can be summarised as follows:

1. Identify all links penetrating a wall and compute their reduced distance q. If the boundary configuration does not change in time, this has to be done only once.
2. Collide on all fluid and boundary nodes. This will provide $f_i^\star(\boldsymbol{x}_{\mathrm{b}}, t)$, $f_{\bar{i}}^\star(\boldsymbol{x}_{\mathrm{b}}, t)$ and $f_i^\star(\boldsymbol{x}_{\mathrm{f}}, t)$.
3. Compute all $f_{\bar{i}}(\boldsymbol{x}_{\mathrm{b}}, t + \Delta t)$ from (11.7).
4. Propagate all remaining populations.
5. Go back to step 1.

The extension of the momentum exchange algorithm to the IBB is straightforward. According to Bouzidi et al. [11], the momentum exchange for a resting boundary link in Fig. 11.5 is

$$\Delta\boldsymbol{p}_i = \Delta x^3 \left(f_i^\star(\boldsymbol{x}_{\mathrm{b}}, t) - f_{\bar{i}}(\boldsymbol{x}_{\mathrm{b}}, t + \Delta t)\right)\boldsymbol{c}_i, \tag{11.8}$$

no matter the value q of the link. An alternative to (11.5), which improves its accuracy, can be found in [17].

11.2.2.2 Moving Boundaries

Most IBB applications in the literature deal with rigid boundaries (which can either be stationary or move on the lattice). There is only a small number of works featuring the IBB for deformable boundaries, e.g. [39]. The immersed boundary method (Sect. 11.4) is a more common approach in those situations. We will therefore not discuss deformable boundaries here.

Lallemand and Luo [40] extended the IBB to moving boundaries. The first ingredient is Ladd's algorithm: the term $-2w_i\rho\boldsymbol{u}_\mathrm{w}\cdot\boldsymbol{c}_i/c_\mathrm{s}^2$ has to be added to the right-hand-side of (11.7), where $\boldsymbol{u}_\mathrm{w}$ is the wall velocity at the intersection point. Also, since there is generally no fluid inside the solid in the IBB framework, a refill mechanism has to be implemented when boundaries move and uncover new (fresh) fluid nodes. We will get back to this in Sect. 11.2.4. Lallemand and Luo's important finding is that the motion of a cylinder on the lattice and the subsequent destruction and creation of fresh nodes leads to some fluctuations of the drag coefficient. This is one of the largest disadvantages of the IBB for moving boundaries; a problem which can be reduced by using an advanced fresh node treatment (Sect. 11.2.4) or the immersed boundary method (Sect. 11.4).

11.2.2.3 Extended and Alternative Methods

Several other second-order accurate bounce-back-based boundary conditions for arbitrary geometries have been proposed. The following list shows a selection of those methods and their most notable properties.

- Yu et al. [41] presented a unified scheme of Bouzidi's algorithm which does not require separate treatment of the regions $q < \frac{1}{2}$ and $q \geq \frac{1}{2}$. Otherwise Yu's and Bouzidi's approaches lead to similar results, including viscosity-dependent slip (not easily solved by TRT/MRT collision models) and violation of mass conservation.
- Ginzburg and d'Humières [17] proposed the so-called *multireflection* boundary condition as an enhanced IBB. The key feature of this method is that it determines the coefficients of the interpolation, rather than heuristically, using the second-order Chapman-Enksog expansion on the interpolated populations. This way, it guarantees the closure condition reproduced by the mesoscopic populations are in exact agreement with the intended hydrodynamic condition. For the reasons explained in Chap. 5, the multireflection method is generally constructed to be formally third-order accurate, and that can be achieved in different ways: the original multireflection scheme [17] adopts a (non-local) interpolation process over five populations, while the more recent MLI scheme [15, 37, 38] only operates over three populations (belonging to the same node) and supplements this information with a (non-local) link-wise second-order finite difference approximation of the hydrodynamic quantity of interest. Both these variants can be implemented in two nodes only [15, 37, 38] and, also in both cases,

the algorithm shall consider a post-collision correction term, which guarantees the method's higher-order accuracy and consistency (i.e. viscosity independence with TRT/MRT collision models). A drawback of the multireflection technique, common to the generality of interpolation-based schemes, is the possible violation of mass conservation.

- Chun and Ladd [18] proposed a method based on the interpolation of the equilibrium distribution. It has the advantage that it requires only one node—in contrast to two or three nodes in the standard linear and quadratic IBB scheme. Similar to the SBB and multireflection schemes, this method guarantees the exact wall location is viscosity-independent with TRT/MRT collision models. This approach is particularly suitable for the time-dependent simulation of geometries with narrow gaps between solids, e.g. for porous media. Mass conservation is generally violated, just as in the majority of interpolation schemes.
- Kao and Yang [36] suggested an interpolation-free method based on the idea of local grid refinement in order to improve the mass conservation of the boundary condition.
- Yin and Zhang [42] presented another improved bounce-back scheme. Their idea was to use Ladd's momentum correction term and linearly interpolate the fluid velocity between a nearby boundary node and the wall location (which can be anywhere between two lattice nodes) to obtain the fluid velocity midway between boundary and solid nodes. This promising method shares common disadvantages with other interpolated bounce-back schemes: violation of mass conservation and viscosity-dependent wall location.

The **interpolated bounce-back (IBB) method** is a common extension of the simple bounce-back scheme for rigid resting or moving obstacles with complex shapes. IBB is second-order accurate but introduces an important weakness: the viscosity-dependent boundary slip is not easily corrected with TRT/MRT collision operators. Furthermore, due to the involved interpolations, IBB is not mass-conserving. Even so, due to its intuitive working principle and relatively simple implementation, the IBB is often the method of choice for improving the SBB accuracy in describing stationary complex geometries.

11.2.3 Partially Saturated Bounce-Back

Now we present the so-called *partially saturated* method (PSM), also known as *grey* LB model or *continuous bounce-back*, where a lattice node can be a pure fluid, a pure solid or a mixed (partially saturated) node as shown in Fig. 11.6. Interestingly,

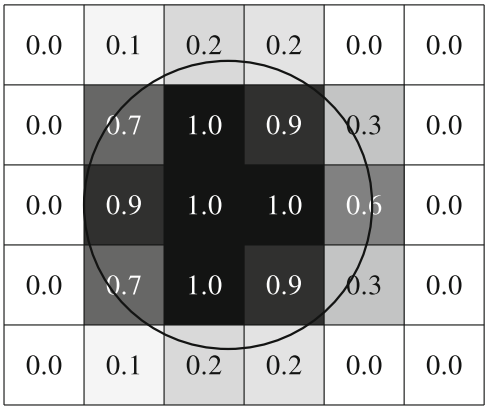

Fig. 11.6 Partially saturated bounce-back. A spherical particle (*circle*) covers a certain amount of each lattice cell. White corresponds to no coverage, black to full coverage. The solid fraction $0 \leq \epsilon \leq 1$ for each cell is shown up to the first digit. Lattice nodes (not shown here) are located at the centre of lattice cells

there exist two research communities which do not seem to interact strongly. The first applies the PSM to simulations of flows in porous media with heterogeneous permeability [43–45]. The other community is interested in suspension flows; they employ the PSM to map the sharp surface of an immersed structure onto the lattice [13, 35, 46, 47]. For the sake of brevity and since both approaches are technically similar, we only elaborate on the latter application. We emphasise that the PSM must not be confused with immersed boundary schemes (Sect. 11.4) which are, according to our definition, fundamentally different in nature. As demonstrated in a series of studies, e.g. [44, 45, 48, 49], the way PSM works can be considered equivalent to the standard LBE with an added friction force. The magnitude of this force varies locally, depending on the nodal fluid/solid fraction. This results in a continuous accommodation of the solution from open (fluid) to very impermeable (solid) regions. Hence, in PSM the nature of the wall can be understood as an interface condition, separating nodes of contrasting properties [48–50].

11.2.3.1 Basic Algorithm

In 1998, Noble and Torczynski [46] presented a bounce-back-based approach, later investigated more thoroughly by Strack and Cook [13], to approximate complex boundaries *on lattice nodes*. The central part of the PSM algorithm is a modified LBGK equation:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + (1 - B)\Omega_i^{\mathrm{f}} + B\Omega_i^{\mathrm{s}} \tag{11.9}$$

where

$$\Omega_i^{\mathrm{f}} = -\frac{f_i(\boldsymbol{x}, t) - f_i^{\mathrm{eq}}(\boldsymbol{x}, t)}{\tau} \tag{11.10}$$

is the standard BGK collision operator for fluid (f) nodes and

$$\Omega_i^{\mathrm{s}} = \left(f_i(\boldsymbol{x},t) - f_{\bar{i}}^{\mathrm{eq}}(\rho,\boldsymbol{u})\right) - \left(f_i(\boldsymbol{x},t) - f_i^{\mathrm{eq}}(\rho,\boldsymbol{u}_{\mathrm{s}})\right) \tag{11.11}$$

is the collision operator for solid (s) nodes. $\boldsymbol{u}$ is the local fluid velocity and $\boldsymbol{u}_{\mathrm{s}}$ is the velocity of the boundary at point $\boldsymbol{x}$. B is a weighting parameter defined by [46]

$$B(\epsilon,\tau) = \frac{\epsilon\left(\tau - \frac{1}{2}\right)}{(1-\epsilon) + \left(\tau - \frac{1}{2}\right)} \tag{11.12}$$

where $0 \leq \epsilon \leq 1$ is the solid fraction of the node. It can be shown that $B(\epsilon)$ increases monotonically between 0 for $\epsilon = 0$ (pure fluid node) and 1 for $\epsilon = 1$ (pure solid node) for any fixed value of $\tau > \frac{1}{2}$. The essential idea is to surrender any shape details of the off-lattice boundary and use an on-lattice volume fraction ϵ instead.

Exercise 11.2 Show that the collision operator in (11.9) is mass-conserving by computing $\sum_i \Omega_i^{\mathrm{f}}$ and $\sum_i \Omega_i^{\mathrm{s}}$.

Note that the PSM assumes the standard BGK form for $B = 0$ and describes a bounce-back of the non-equilibrium for $B = 1$. A mixed collision and bounce-back scheme is performed for partially saturated nodes ($0 < \epsilon < 1$), which are only found in direct boundary neighbourhood (Fig. 11.6).

Force and torque acting on the boundary can be computed from

$$\begin{aligned} \boldsymbol{f} &= \frac{\Delta x^3}{\Delta t} \sum_{\boldsymbol{x}_n} B(\boldsymbol{x}_n) \sum_i \Omega_i^{\mathrm{s}}(\boldsymbol{x}_n)\boldsymbol{c}_i, \\ \boldsymbol{T} &= \frac{\Delta x^3}{\Delta t} \sum_{\boldsymbol{x}_n} B(\boldsymbol{x}_n)(\boldsymbol{x}_n - \boldsymbol{R}) \times \sum_i \Omega_i^{\mathrm{s}}(\boldsymbol{x}_n)\boldsymbol{c}_i, \end{aligned} \tag{11.13}$$

respectively, where the $\boldsymbol{x}_n$ are all lattice nodes in contact with the solid (including all interior nodes), i.e. those nodes with $\epsilon > 0$, i runs over all lattice directions at a given position $\boldsymbol{x}_n$ and $\boldsymbol{R}$ is the location of the centre of mass of the solid. Updating the solid's momentum and angular momentum according to (11.13) guarantees overall momentum conservation.

It is worth mentioning that Zhou et al. [51] have combined the node-based method with Lees-Edwards BCs, which is relevant for the simulation of large bulk systems. Furthermore, Chen et al. [47] have recently proposed a combination of the PSM and a ghost method (Sect. 11.3) to improve the no-slip condition at the boundary surface. Yu et al. [52] proposed another variant taking into account a mass-conserving population migration process in the vicinity of moving walls.

11.2.3.2 Advantages and Limitations

The implementation of this algorithm is relatively straightforward. If the boundary is stationary, as for example encountered for a porous medium, ϵ can be computed once at each lattice site. For moving boundaries, ϵ has to be updated, which is the most challenging aspect of the PSM. Also, the correspondence between ϵ and the actual boundary shape is not trivial and requires some calibration [44, 45, 49]. For example, Han and Cundall [35] use a sub-cell method to estimate ϵ for a given lattice site while Chen et al. [47] employ a cut-cell approach. Apart from updating ϵ, no additional measures have to be taken when objects are moving on the lattice. In particular, fresh nodes appearing on the rear of a moving obstacle do not have to be treated in a special way. Neither are fluid nodes destroyed when they are covered by the advancing boundary. Since the interior fluid is never destroyed and still participates in the collisions described by Ω_i^s, mass and momentum are conserved.

In reality, curved boundaries in the PSM are nothing more than a sophisticated staircase (cf. Fig. 11.6). In the PSM, there is no information about the distance between lattice nodes and boundaries; instead, the local fluid filling fraction is considered. It is easy to imagine that many different boundary configurations can lead to the same filling fraction. Therefore, the PSM fails to capture the correct shape of the boundary.

Strack and Cook [13] performed careful 3D benchmark tests of the PSM. The authors report a significantly smoother motion when the weight $B(\epsilon, \tau)$ is used, rather than just falling back to a staircase approximation of the boundary. This is mostly due to the smooth uncovering of fluid nodes which have previously been solid nodes and *vice versa*. However, the smoothness of the observables (velocity, force and torque) depends on the accurate computation of the solid ratio ϵ.

Later, Han and Cundall [35] investigated the resolution sensitivity of the PSM and Ladd's BB scheme (Sect. 11.2.1) in 2D. They found that both methods are comparably accurate in terms of the drag coefficient of relatively large circles (diameter $\approx 10\Delta x$). However, for diameters as small as $4 - 5\Delta x$, the PSM is superior, in particular when the objects are moving on the lattice.

The PSM has a number of advantages. The first is that one does not face the fresh node problem (Sect. 11.2.4) which causes some trouble in most of the other BB variants. Moreover, the PSM, unlike IBB, is exactly mass conserving. Another advantage is the absence of interpolations to enforce the boundary condition. This makes the PSM a promising candidate for dense suspensions and porous media with small pore sizes.

However, when used for suspension flow, the PSM has so far mostly been applied to very simple geometries like circles in 2D or spheres in 3D. Although it is possible to construct more complex geometries by assembling several circles or spheres [35], additional work is necessary to make the PSM more attractive for moving, arbitrarily shaped boundaries. (Chen et al. [47] provide a short discussion of algorithms which can be used for more complex shapes.) Also, by sacrificing the treatment of the exact boundary shape, one cannot expect that the no-slip condition is accurately satisfied at the boundary [48, 49]. More investigations of the accuracy of the PSM for simple

and complex boundary shapes would certainly be beneficial. Furthermore, the PSM in its present form is not suitable for the simulation of deformable boundaries or thin shells with fluid on both sides (unlike interpolated bounce-back, for example).

The **partially saturated method (PSM)** is a node-based method. PSM is exactly mass-conserving and does not require the treatment of fresh fluid nodes. The disadvantage is the difficulty of finding correct values for the solid fraction near solid boundaries, which is effectively limiting this method to stationary geometries (where the solid fraction has to be computed only once) or to spherical particles.

11.2.4 Destruction and Creation of Fluid Nodes

When boundaries move, it happens from time to time that lattice nodes cross the boundary, either from the fluid to the inside of the boundary or *vice versa*. If the interior of the boundary is not filled with a fluid, the former event requires the *destruction*, the latter the *creation* of a fluid site as shown in Fig. 11.7. Newly created nodes are also called *fresh nodes*. This applies to most methods described in Sect. 11.2 and also Sect. 11.3, but not to Ladd's method (Sect. 11.2.1) or the partially saturated method (Sect. 11.2.3) where nodes are neither created nor destroyed. Generally the number of fluid and solid nodes is not conserved when boundaries move on the lattice.

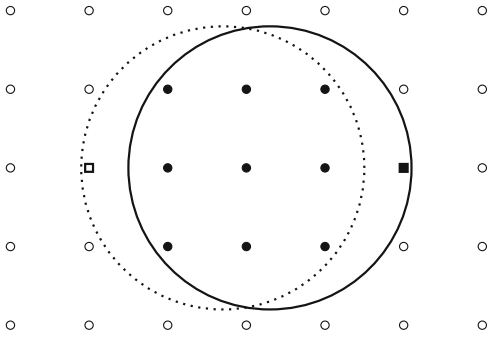

Fig. 11.7 Creation and destruction of fluid nodes. A particle is moving from its previous position (*dotted circle*) to its new position (*solid circle*). As a consequence, one fresh fluid node (*open square*) appears behind the particle and a fluid node is destroyed (*solid square*) at the front of the particle. Fluid and solid nodes are shown as *open and solid symbols*, respectively

Destruction is straightforward [27]: the state of the site is switched from "fluid" to "solid", and its momentum and angular momentum are transferred to the solid. For a destroyed fluid node at $\boldsymbol{x}_\mathrm{d}$ with density ρ and velocity $\boldsymbol{u}$, the particles receives a momentum contribution $\rho \boldsymbol{u} \Delta x^3$ and an angular momentum contribution $(\boldsymbol{x}_\mathrm{d} - \boldsymbol{R}) \times (\rho \boldsymbol{u}) \Delta x^3$ where $\boldsymbol{R}$ is the particle's centre of mass. Finally, the fluid information of the site at $\boldsymbol{x}_\mathrm{d}$ is omitted.

The inverse process, creation of a fluid site, is more difficult because the density, velocity and even all populations are unknown at first. The simplest approach to initialise a fresh node at point $\boldsymbol{x}_\mathrm{f}$ and time t is to estimate the density as the average of the neighbouring fluid sites [27],

$$\rho_\mathrm{f} = \rho(\boldsymbol{x}_\mathrm{f}, t) = \frac{1}{N_\mathrm{f}} \sum_i \rho(\boldsymbol{x}_\mathrm{f} + \boldsymbol{c}_i \Delta t, t), \tag{11.14}$$

where the sum runs only over those N_f neighbouring sites which are fluid. The velocity $\boldsymbol{u}_\mathrm{f}$ of the fresh node is computed from the known boundary velocity of the obstacle at the same point *via* (11.4). The populations f_i are then initialised with their equilibrium values $f_i^\mathrm{eq}(\rho_\mathrm{f}, \boldsymbol{u}_\mathrm{f})$. As additional step, the momentum and angular momentum of the fresh node have to be subtracted from the solid.

Although this approach is easy to implement, two disadvantages are obvious: the total mass in the system is generally not conserved, and the non-equilibrium contribution of the fresh node is missing, which can lead to distortions of the flow field.

Chen et al. [14] compared the above-mentioned approach and three other algorithms to initialise fresh nodes. One of those relies on extrapolation of populations from neighbouring fluid sites [40]. The other two approaches, both first described in [14], are based on the consistent initialisation [53] (see also Sect. 5.5). From benchmark tests involving moving cylinders, the authors come to the conclusion that the consistent initialisation methods are usually more accurate than interpolation [27] or extrapolation [40].

The **destruction and creation of fluid nodes** is necessary when the standard or interpolated bounce-back method (or certain other boundary conditions) are used for moving boundaries. Boundary treatments without the need for this consideration are Ladd's method for suspended particles and the partially saturated method. It has been observed that the treatment of fresh nodes is crucial to reduce oscillations of the particle drag and creation of detrimental sound waves.

11.2.5 Wall Shear Stress

We conclude this section with a more general discussion of the wall shear stress that is useful for most LB boundary conditions.

Several diseases of the circulatory system are assumed to be linked to pathological levels or changes of the shear stress at the arterial wall (see, e.g., [54] and references therein). Two prominent examples are atherosclerosis or aneursym formation. In recent years, a growing number of scientists became interested in the LB modelling of blood flow in realistic blood vessel geometries. Apart from finding and implementing reasonable boundary conditions, a key question is how the wall shear stress (WSS) can be computed and how accurate the obtained values are.

We will provide a brief review of the comparatively small number of publications in this field, but before that a few words about the WSS are necessary. The WSS is tightly connected to the momentum exchange at the boundary. Evaluating (11.2) is not sufficient to find the WSS, though. The reason is that WSS is a local quantity and not an integrated property of the entire surface of the boundary.

In order to find the WSS, the boundary location *and* orientation have to be known at each point of interest. Assuming that $\boldsymbol{x}_\mathrm{w}$ is a point on the wall, we denote $\hat{\boldsymbol{n}}$ the wall normal vector at $\boldsymbol{x}_\mathrm{w}$ pointing inside the fluid domain. For a given fluid stress tensor $\boldsymbol{\sigma}$ at $\boldsymbol{x}_\mathrm{w}$, we first define the *traction* vector as $\boldsymbol{T} = \boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}}$. It is the force acting on an infinitesimal, oriented wall area element $\mathrm{d}\boldsymbol{A} = \mathrm{d}A\,\hat{\boldsymbol{n}}$. The WSS vector $\boldsymbol{\tau}$ is the tangential component of the traction vector, i.e. we have to subtract the normal component of the traction:

$$\boldsymbol{\tau} = \boldsymbol{T} - (\boldsymbol{T} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}, \quad \tau_\alpha = \sigma_{\alpha\beta}\hat{n}_\beta - (\sigma_{\beta\gamma}\hat{n}_\beta\hat{n}_\gamma)\hat{n}_\alpha. \tag{11.15}$$

The subtracted normal component $(\boldsymbol{T} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$ contributes to the wall pressure. It is common to report only the magnitude of the WSS vector, simply called the WSS $\tau = |\boldsymbol{\tau}|$ (not to be confused with the BGK relaxation time).

We distinguish three typical situations:

1. The normal vector $\hat{\boldsymbol{n}}$ is known everywhere on the boundary, but the geometry is approximated by a staircase boundary (simple bounce-back).
2. The geometry is only known as a staircase, and no additional information about the normal vector $\hat{\boldsymbol{n}}$ is available.
3. A higher-order boundary conditions is used for the boundaries (e.g. IBB). We will not consider this case here.

An example for the first case is the discretisation of a known boundary geometry where a surface tessellation is converted to a staircase surface. Here we still have access to the original normal vectors, but the LB simulation is only aware of the staircase. The second case is important when voxel data is directly converted into a staircase geometry, without *a priori* knowledge of the boundary normals. This means that we first need to estimate $\hat{\boldsymbol{n}}$ from the known data before we can compute the WSS.

Stahl et al. [55] were the first authors to provide a careful investigation of the behaviour of the WSS in LB simulations with staircase geometries. They presented a scheme to obtain the unknown normal vectors $\hat{\boldsymbol{n}}$ from the flow field. This is straightforward in 2D where the no-penetration condition requires $\boldsymbol{u} \cdot \hat{\boldsymbol{n}} = 0$ for the fluid velocity $\boldsymbol{u}$ at the wall. From the known velocity $\boldsymbol{u}$ we can easily compute the unknown $\hat{\boldsymbol{n}}$ up to its sign. This is more complicated but possible in 3D, where additional information is required (see [55] for details). The authors then investigated the accuracy of the stress computation near the boundary in an inclined Poiseuille flow. They found strongly anisotropic behaviour: the error is minimum for walls aligned with one of the major lattice directions (i.e. when the wall is flat), but larger errors for arbitrarily inclined walls. This error, however, decreases when the stress is evaluated farther away from the wall. Therefore the authors suggested to measure the fluid stress a few lattice sites away from the wall. All in all, it requires relatively high resolutions (several $10\Delta x$) to estimate the WSS reasonably well in a staircase geometry, which makes this method unfeasible when the average channel diameter is small.

Later, Matyka et al. [56] proposed a different scheme to obtain a better estimate of the unknown normal vectors $\hat{\boldsymbol{n}}$. Their idea was to compute a weighted average of the staircase information of the neighbouring lattice nodes. The advantage of their approach is that it is based on the geometry alone; it is independent of the flow field. Furthermore, the authors showed that the WSS error is dominated by the flow field error and not by an inaccurate approximation of the normal vector. The flow field error in turn is caused by the staircase approximation (modelling error), which can only be decreased by using a more accurate boundary condition for the LBM.

Pontrelli et al. [54] used a finite-volume LBM to compute the WSS in a small artery with a realistic endothelial wall profile. Unfortunately the authors did not provide a benchmark test of the WSS accuracy in their setup. It would be interesting to investigate whether the finite-volume LBM is able to mitigate the shortcomings of the regular lattice with staircase approximation.

Very recently, Kang and Dun [57] studied the accuracy of the WSS in inclined 2D Poiseuille and Womersley flows for BGK and MRT collision operators and for SBB and IBB at the walls. One of the basic results is that, in channels *aligned* with a major lattice axis, the WSS converges with a first-order rate upon grid refinement when it is evaluated at the fluid layer closest to the wall and with a second-order rate when the result is extrapolated to the wall location. This is no surprise since the distance of the last fluid layer and the wall itself converges to zero with first-order rate. The authors report similar results for BGK and MRT for their chosen parameter range. When the flow in an *inclined* channel is simulated, the choice of the boundary condition plays a significant role. IBB leads to errors which are about one order of magnitude smaller than for SBB. Moreover, MRT leads to slightly better results than BGK.

The choice of wall boundary condition critically affects the quality of **wall shear stress** estimates. The best results are obtained when a curved boundary condition is used as this increases the accuracy of the flow field and the fluid stress tensor in direct vicinity of the wall. Further research is necessary to develop improved boundary conditions for more accurate WSS computations in complex geometries.

11.3 Ghost Methods

We will now present LB boundary methods which require extrapolations. A typical scenario is the extrapolation of fluid properties at virtual nodes within a solid body. These so-called *ghost* nodes then participate in collision and propagation like normal fluid nodes. This process produces those populations which stream out of the solid and would otherwise be unknown. After providing some definitions in Sect. 11.3.1, we discuss three distinct classes of extrapolation-based boundary conditions: the Filippova-Hänel and Mei-Luo-Shyy methods (Sect. 11.3.2), the Guo-Zheng-Shi method (Sect. 11.3.3) and comparably novel image-based ghost methods (Sect. 11.3.4). Some recommended articles are [58–62].

11.3.1 Definitions

In order to understand the motivation for the boundary conditions presented in this section, we first have to define certain terms and understand their implications.

- *Extrapolation* in the present context means that the known information of a quantity (e.g. velocity) within a geometrical region is used to approximate the quantity outside this region. The known region is typically the fluid region whereas the interior of a solid is unknown. For example, if we know the velocity u at points x and $x' = x + \Delta x$ (which may be inside the fluid) but not at $x'' = x' + \Delta x'$ (which may be inside the solid), we can still approximate it by *assuming* a linear behaviour and write

$$u(x'') = u(x') + \frac{u(x') - u(x)}{\Delta x} \Delta x'. \tag{11.16}$$

  Higher-order extrapolations require more known data points (usually $n+1$ points for extrapolation of order n). Extrapolations can often lead to instability (in particular when Δx in (11.16) is small) and loss of accuracy.

- *Fictitious domain* methods are based on the idea that the solution of a problem in a given (usually complex) domain Ω can be simpler when instead a substitute problem in a larger (and simpler) domain Ω' with $\Omega \subset \Omega'$ is solved. This obviously means that information in the complement region $\Omega' \setminus \Omega$ has to be constructed. This usually involves extrapolations.
- *Ghost* methods are a special case of fictitious domain methods where virtual fluid nodes (ghost nodes) are created inside the solid region close to the boundary. Extrapolations of fluid and boundary properties are used to reconstruct the ghost nodes. These nodes then participate in collision and propagation in the normal way in order to supply the boundary nodes with otherwise missing populations. Unfortunately these methods are sometimes denoted as *sharp-interface immersed-boundary methods*, although they hardly share any similarities with the immersed boundary method originally introduced by Peskin (Sect. 11.4).

Revisiting the bounce-back boundary conditions presented in Sect. 11.2, we can make the following comments:

- Standard and interpolated bounce-back do not involve any extrapolation or ghost nodes. Although one can implement both methods with the help of nodes which are on the solid side of the boundary, these nodes are only used for memory storage purposes and do not qualify these methods as ghost methods.
- The partially saturated method uses nodes which are inside the solid, but no extrapolations are required to create them. The reason is that the fluid is simply kept in the interior without the need to reconstruct it at every time step.

Therefore, neither of the bounce-back-based boundary conditions in Sect. 11.2 is an extrapolation or ghost method.

We present three extrapolation-based methods in more detail: the Filippova-Hänel and Mei-Luo-Shyy methods, the Guo-Zheng-Shi method and image-based ghost methods. Apart from this, the method by Verschaeve and Müller [63] as an extension of [64] to curved boundaries is yet another alternative which we will, however, not discuss in detail. In short, the underlying idea of [63] is to have boundary nodes in both the fluid and solid regions and to interpolate and extrapolate fluid properties, respectively. The equilibrium distributions are reconstructed from the density and velocity, the non-equilibrium distributions from the velocity gradient.

11.3.2 Filippova-Hänel (FH) and Mei-Luo-Shyy (MLS) Methods

In 1998, Filippova and Hänel [65] proposed the first LB boundary condition for curved geometries (FH method) using extrapolations. They assume the following situation: a population $f_i^\star(\boldsymbol{x}_\mathrm{b}, t)$ propagates towards a wall located between a boundary node at $\boldsymbol{x}_\mathrm{b}$ and a solid node at $\boldsymbol{x}_\mathrm{s} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{c}_i \Delta t$. The wall is located at

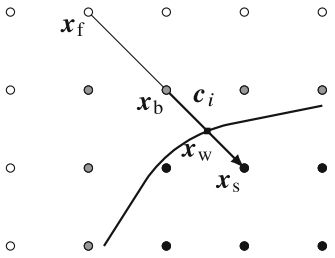

Fig. 11.8 Filippova and Hänel boundary condition. A link $\boldsymbol{c}_i$ is cut by a curved boundary at $\boldsymbol{x}_w$ (*solid square*). Fluid, boundary and solid nodes are shown as *white, grey and black circles*, respectively. Information about the velocity at the solid node $\boldsymbol{x}_s$ is required to find the post-streaming population $f_{\bar{i}}(\boldsymbol{x}_b)$. The original method [65] requires the velocity at $\boldsymbol{x}_b$ only, while the improved method [58] uses the velocity at $\boldsymbol{x}_f$ as well

$\boldsymbol{x}_w = \boldsymbol{x}_b + q\boldsymbol{c}_i\Delta t$ as illustrated in Fig. 11.8. The question is how to find the missing population $f_{\bar{i}}(\boldsymbol{x}_b, t + \Delta t)$.

11.3.2.1 Original Method by Filippova and Hänel (FH)

Filippova and Hänel [65] suggested the equation

$$f_{\bar{i}}(\boldsymbol{x}_b, t + \Delta t) = (1 - \chi) f_i^\star(\boldsymbol{x}_b, t) + \chi f_i^{\mathrm{eq}}(\boldsymbol{x}_s, t) - 2w_i \frac{\boldsymbol{u}_w \cdot \boldsymbol{c}_i}{c_s^2} \tag{11.17}$$

for each direction $\boldsymbol{c}_i$ crossing a wall. Here, $\boldsymbol{u}_w$ is the wall velocity (i.e. the velocity of the wall at the intersection point $\boldsymbol{x}_w$, cf. Fig. 11.8) and χ a weighting factor (with the dimensionless BGK relaxation time τ):

$$\chi = \begin{cases} \frac{1}{\tau}\frac{2q-1}{1-\frac{1}{\tau}} & q < \frac{1}{2} \\ \frac{1}{\tau}(2q-1) & q \geq \frac{1}{2} \end{cases}. \tag{11.18}$$

Exercise 11.3 Show that (11.17) reduces to simple bounce-back for $q = \frac{1}{2}$.

Equation (11.17) is essentially an interpolation of populations at $\boldsymbol{x}_b$ and $\boldsymbol{x}_s$, but we still have to investigate the shape of the required equilibrium term $f_i^{\mathrm{eq}}(\boldsymbol{x}_s, t)$. Filippova and Hänel [65] construct the "equilibrium distribution in the rigid nodes" from

$$f_i^{\mathrm{eq}}(\boldsymbol{x}_s, t) = w_i \left(\frac{p(\boldsymbol{x}_b, t)}{c_s^2} + \frac{\boldsymbol{c}_i \cdot \boldsymbol{u}_s}{c_s^2} + \frac{(\boldsymbol{c}_i \cdot \boldsymbol{u}_b)^2}{2c_s^4} - \frac{\boldsymbol{u}_b \cdot \boldsymbol{u}_b}{2c_s^2} \right). \tag{11.19}$$

where we have used the abbreviations $\boldsymbol{u}_b = \boldsymbol{u}(\boldsymbol{x}_b, t)$ and $\boldsymbol{u}_s = \boldsymbol{u}(\boldsymbol{x}_s, t)$. This is nearly the standard incompressible equilibrium evaluated at $\boldsymbol{x}_b$, with the only exception that

the fluid velocity $\boldsymbol{u}_\mathrm{b}$ is replaced by the solid node velocity $\boldsymbol{u}_\mathrm{s}$ in the linear term. The authors suggested

$$\boldsymbol{u}_\mathrm{s} = \begin{cases} \boldsymbol{u}_\mathrm{b} & q < \frac{1}{2} \\ \frac{q-1}{q}\boldsymbol{u}_\mathrm{b} + \frac{1}{q}\boldsymbol{u}_\mathrm{w} & q \geq \frac{1}{2} \end{cases} \tag{11.20}$$

to find the missing velocity at $\boldsymbol{x}_\mathrm{s}$.

This deserves a few comments:

- For $q \geq \frac{1}{2}$, the solid node velocity is obtained by *extrapolating* the velocity at $\boldsymbol{x}_\mathrm{s}$ from $\boldsymbol{x}_\mathrm{b}$ and $\boldsymbol{x}_\mathrm{w}$.
- Since the extrapolation would lead to unstable results for $q \to 0$, the authors fall back to $\boldsymbol{u}_\mathrm{s} = \boldsymbol{u}_\mathrm{b}$ for $q < \frac{1}{2}$.
- The choice of the incompressible equilibrium also explains why the fluid density does not appear in the momentum exchange term on the right-hand-side of (11.17): in the incompressible method the density is constant and typically set to unity.

Although the FH method reduces to simple bounce-back for $q = \frac{1}{2}$, it is conceptually different from interpolated bounce-back (IBB, Sect. 11.2.2) which also reduces to simple bounce-back for $q = \frac{1}{2}$. For any q-value, only one fluid node is required in (11.17), which makes the FH method more local than IBB. The FH method requires an extrapolation for $q \geq \frac{1}{2}$, IBB does not.

11.3.2.2 Improvements by Mei, Luo and Shyy (MLS)

The FH method has the major disadvantage that the weight χ diverges for $\tau \to 1$ and $q < \frac{1}{2}$, which leads to instability. Mei et al. [58] therefore analysed the FH method and its stability properties in detail and proposed an improved version (MLS method). The starting point for the improvement is to realise that there are different ways to construct the term $f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t)$. The authors proposed new expressions for $q < \frac{1}{2}$:

$$\boldsymbol{u}_\mathrm{s} = \boldsymbol{u}_\mathrm{f}, \quad \chi = \frac{2q-1}{\tau - 2}, \tag{11.21}$$

where $\boldsymbol{u}_\mathrm{f} = \boldsymbol{u}(\boldsymbol{x}_\mathrm{f}, t)$ and $\boldsymbol{x}_\mathrm{f} = \boldsymbol{x}_\mathrm{b} - \boldsymbol{c}_i \Delta t$ is the location of the fluid node beyond the boundary node (cf. Fig. 11.8). The expressions for $q \geq \frac{1}{2}$ remain untouched.

Mei et al. [58] showed that this modification indeed improves the stability of the original FH method, but they are also sacrificing its locality as a boundary and a fluid node are required. The authors further mention that the above expressions are only strictly valid for stationary flows. They therefore suggested a higher-order extrapolation at $\boldsymbol{u}_\mathrm{s}$ for transient flows.

It is important to note that most follow-up works in the literature employ the improved [58] rather than the original [65] implementation. In the following we summarise some notable progress:

- Mei et al. [66] were the first to perform a thorough comparative evaluation of the momentum exchange algorithm (MEA) and stress integration in the context of the MLS method to obtain drag and lift coefficients at stationary curved boundaries. They found that the stress integration is much more demanding in terms of implementation effort and computing time while the MEA is still relatively accurate. Mei et al. therefore recommend to use the MEA.
- Like other interpolation- and extrapolation-based approaches, the FH and MLS methods suffer from a violation of mass conservation. Therefore, Bao et al. [67] analysed the mechanism responsible for the mass leakage in those boundary treatments and presented an improved mass-conserving method.
- Wen et al. [59] extended the MEA [66] to moving boundaries.

11.3.3 Guo-Zheng-Shi (GZS) Method

Guo et al. [60] proposed yet another extrapolation-based LB boundary condition for curved boundaries (GZS). The problem is the same as in Sect. 11.3.2 and Fig. 11.8. In particular, the cut link $\boldsymbol{c}_i$ points into the solid.[3]

The question is how to find $f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t + \Delta t)$. The GZS method uses a fictitious fluid node at $\boldsymbol{x}_\mathrm{s}$ which is assigned an equilibrium value

$$f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\rho_\mathrm{s}, \boldsymbol{u}_\mathrm{s}) \tag{11.22}$$

where $f_i^{\mathrm{eq}}(\rho, \boldsymbol{u})$ is the standard incompressible equilibrium. Note that the only ficticious nodes are those solid nodes which are directly connected to a boundary node by a lattice vector $\boldsymbol{c}_i$.

The authors approximate the density at the solid site by its neighbour value: $\rho_\mathrm{s} = \rho(\boldsymbol{x}_\mathrm{s}, t) = \rho(\boldsymbol{x}_\mathrm{b}, t)$. For the velocity, they suggested

$$\boldsymbol{u}_\mathrm{s} = \begin{cases} q\boldsymbol{u}^{(1)} + (1-q)\boldsymbol{u}^{(2)} & q < \frac{3}{4} \\ \boldsymbol{u}^{(1)} & q \geq \frac{3}{4} \end{cases} \tag{11.23}$$

[3] Guo et al. [60] defined $\boldsymbol{c}_i$ exactly the other way around.

where $\boldsymbol{u}^{(1)}$ and $\boldsymbol{u}^{(2)}$ are extrapolations using the nodes at $\boldsymbol{x}_\mathrm{b}$ and $\boldsymbol{x}_\mathrm{f}$, respectively:

$$\begin{aligned}\boldsymbol{u}^{(1)} &= \frac{(q-1)\boldsymbol{u}_\mathrm{b} + \boldsymbol{u}_\mathrm{w}}{q},\\ \boldsymbol{u}^{(2)} &= \frac{(q-1)\boldsymbol{u}_\mathrm{f} + 2\boldsymbol{u}_\mathrm{w}}{1+q}.\end{aligned} \tag{11.24}$$

This means that $\boldsymbol{u}_\mathrm{s}$ can be different for each considered link $\boldsymbol{c}_i$ crossing a boundary; it is therefore not a property of the position $\boldsymbol{x}_\mathrm{s}$ alone. When q is large enough, an extrapolation from the closest fluid node at $\boldsymbol{x}_\mathrm{b}$ is sufficiently stable, but for smaller q-values an extrapolation from the fluid node at $\boldsymbol{x}_\mathrm{f}$ becomes necessary.

Now, apart from the equilibrium, the GZS method also involves the non-equilibrium populations at the solid node. The authors proposed the extrapolation

$$f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t) = \begin{cases} q f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{b}, t) + (1-q) f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{f}, t) & q < \frac{3}{4} \\ f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{b}, t) & q \geq \frac{3}{4} \end{cases}. \tag{11.25}$$

The GZS algorithm includes the following steps:

1. Find q for a cut link connecting a boundary node $\boldsymbol{x}_\mathrm{b}$ and a solid node $\boldsymbol{x}_\mathrm{s}$.
2. Reconstruct the populations of the fictitious nodes by

$$f_{\bar{i}}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) + f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t), \tag{11.26}$$

   where the equilibrium and non-equilibrium parts are computed from (11.22) and (11.25).
3. Collide on all fluid/boundary nodes *and* fictitious nodes.[4]
4. Stream populations from all fluid/boundary nodes *and* fictitious nodes to their fluid neighbours. In particular, $f_{\bar{i}}^{\star}$ streams from the fictitious to the boundary node and provides the missing value for $f_i(\boldsymbol{x}_\mathrm{b}, t + \Delta t)$.
5. Go back to step 1.

According to Guo et al. [60], the present method has advantages over the methods in Sect. 11.3.2. First, while the FH and MLS methods assume a slowly varying flow field, the GZS method only requires a low Mach number flow which can be unsteady. Secondly, the GZS scheme is more stable than the MLS approach.

We would also like to mention that Guo et al. [60] view FH and MLS as improved bounce-back methods. Although that statement is certainly not wrong (the functional form for the missing population $f_{\bar{i}}(\boldsymbol{x}_\mathrm{b}, t+\Delta t)$ is similar to the standard and interpolated bounce-back expressions), the FH, MLS and GZS methods all require

[4] In the original paper [60], the fictitious populations are already constructed in their post-collision state $f_{\bar{i}}^{\star}(\boldsymbol{x}_\mathrm{s}, t) = f_{\bar{i}}^{\mathrm{eq}}(\boldsymbol{x}_\mathrm{s}, t) + (1 - \frac{1}{\tau}) f_{\bar{i}}^{\mathrm{neq}}(\boldsymbol{x}_\mathrm{s}, t)$. In this case, collision on fictitious nodes is of course not additionally performed.

extrapolations and are ghost-like methods. They are therefore conceptually different from the bounce-back methods presented in Sect. 11.2 which are all extrapolation-free.

11.3.4 Image-Based Ghost Methods

Only in 2012, Tiwari and Vanka [61] developed a ghost-fluid boundary condition for the LBM which is based on the so-called *image* method. The idea of their boundary condition is illustrated in Fig. 11.9.

The algorithm consists of the following steps:

1. Identify all required ghost nodes $\boldsymbol{x}_\mathrm{s}$. Those are all solid nodes which are connected to at least one boundary node along a lattice vector $\boldsymbol{c}_i$.
2. For each ghost node $\boldsymbol{x}_\mathrm{s}$ find the closest point $\boldsymbol{x}_\mathrm{w}$ on the wall. We define $\boldsymbol{n} = \boldsymbol{x}_\mathrm{w} - \boldsymbol{x}_\mathrm{s}$ as the outward-pointing normal vector at the wall. Note that $|\boldsymbol{n}|$ is the distance of the ghost node from the wall and $\boldsymbol{n}$ is generally not aligned with any of the lattice vectors $\boldsymbol{c}_i$.
3. The next step is to find the *image point* $\boldsymbol{x}_\mathrm{i}$ in the fluid:

$$\boldsymbol{x}_\mathrm{i} = \boldsymbol{x}_\mathrm{s} + 2\boldsymbol{n} = \boldsymbol{x}_\mathrm{w} + \boldsymbol{n}. \tag{11.27}$$

   For a stationary boundary, steps 1–3 have to be performed only once.
4. Interpolate the required fluid properties (velocity and density) at the image point $\boldsymbol{x}_\mathrm{i}$ to obtain $\boldsymbol{u}_\mathrm{i}$ and ρ_i. The interpolation process is somewhat tedious as it depends on whether interpolation support points are located in the fluid or on the wall. We refer to [61] for a detailed discussion.

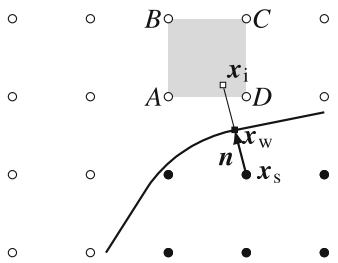

Fig. 11.9 Image-based ghost method. For each ghost node $\boldsymbol{x}_\mathrm{s}$, the closest wall point $\boldsymbol{x}_\mathrm{w}$ is computed. The corresponding image point $\boldsymbol{x}_\mathrm{i}$ in the fluid (*open circles*) is constructed along the normal vector $\boldsymbol{n}$. Fluid properties at the image point are obtained from an interpolation in the grey region (fluid nodes A, B, C, D). A different interpolation is required if not all interpolation support points are located within the fluid

5. Extrapolate velocity and density along the normal $\boldsymbol{n}$ to the ghost node. Tiwari and Vanka [61] used

$$\boldsymbol{u}_\mathrm{s} = 2\boldsymbol{u}_\mathrm{w} - \boldsymbol{u}_\mathrm{i}, \quad \rho_\mathrm{s} = \rho_\mathrm{i}. \tag{11.28}$$

   The first equation refers to a linear extrapolation of the velocity, the second to a zero density gradient at the wall.
6. Compute the equilibrium distributions at the ghost nodes from $f_i^\mathrm{eq}(\boldsymbol{x}_\mathrm{s}, t) = f_i^\mathrm{eq}(\rho_\mathrm{s}, \boldsymbol{u}_\mathrm{s})$.
7. The non-equilibrium distributions $f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{s}, t)$ are obtained like the fluid density: interpolate them at $\boldsymbol{x}_\mathrm{i}$ first, then apply $f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{s}, t) = f_i^\mathrm{neq}(\boldsymbol{x}_\mathrm{i}, t)$.
8. Combine the equilibrium and non-equilibrium distributions at the ghost nodes and perform the propagation step, followed by the collision step.
9. Go back to step 1.

This algorithm deserves a few remarks:

- In contrast to the previous methods in this section, $\boldsymbol{u}_\mathrm{s}$ does not depend on the considered link $\boldsymbol{c}_i$; it is rather a unique property of each ghost node.
- According to [61], the extrapolated values of density, velocity and non-equilibrium distributions are post-collision rather than pre-collision. This is unusual since the moments (density, velocity, stress) are normally computed after the previous streaming and before the next collision step.
- The boundary condition is based on the hydrodynamic fields rather than the populations. This allows implementing Neumann boundary conditions. The authors for example demonstrated the applicability of their method for inlet and outlet boundary conditions [61].
- Extrapolation along normal vectors as in step 5 avoids typical stability issues encountered with other extrapolation methods.
- Although being trivial for circular or spherical boundary segments, finding the image point can be tedious for complex boundary shapes. Also the interpolation at the image points is complicated if not all interpolation support points are within the fluid domain. The application of this boundary condition to moving boundaries of complex shape, in particular in 3D, is therefore difficult and expensive. Tiwari and Vanka [61] simulated only circular boundaries in 2D.
- The assumption of a zero density gradient across the boundary is a gross over-simplification. For example, it fails when a force density along the extrapolation direction exists which is balanced by a pressure gradient [62]. Since errors in the pressure gradient are of higher order, the velocity profile may still be second-order accurate, though. A similar objection can be made for the non-equilibrium distributions. At least a linear extrapolation for the density and the non-equilibrium distributions are required to accurately capture second-order flows like the Poiseuille flow.

Several extensions and improvements of the algorithm have been proposed in the meantime:

- Khazaeli et al. [68] followed a similar route as Tiwari and Vanka [61] to impose higher-order boundary conditions for coupled fluid-heat problems in the two-population LBM.
- Mohammadipoor et al. [62] followed the same line as [61] and extended the approach of Zou and He [69] to curved boundaries.
- Pellerin et al. [70] proposed an image-based method that relies only on equilibrium distributions.

There exist several **extrapolation-based** boundary conditions for the LBM. These methods are conceptually more difficult than bounce-back-based methods. A common algorithmic complication all these methods share with the interpolated bounce-back method is the detection of boundary points (either on cut links or closest points to ghost nodes). In practice, these methods are quite unhandy for moving boundaries of complicated shape although they are promising candidates for highly accurate boundary conditions when properly applied. More research is required to make extrapolation-based method more attractive for moving objects with non-trivial shape.

11.4 Immersed Boundary Methods

The immersed boundary method (IBM) [71–73] is older than LBM, but the combination of both was not suggested before 2004 [74] (Sect. 11.4.1). The basic idea of the IBM is to approximate a boundary by a set of off-lattice marker points that affect the fluid only *via* a force field. An interpolation stencil is introduced to couple the lattice and the marker points (Sect. 11.4.2). This allows a relatively simple implementation of complex boundaries. There are several IBM variants, for example explicit (Sect. 11.4.3) or direct-forcing (Sect. 11.4.4) for rigid boundaries and explicit IBM for deformable boundaries (Sect. 11.4.5). We also mention a series of other related boundary conditions which are less commonly used (Sect. 11.4.6). We recommend reading [75, 76] for introductions and investigations of the IBM in conjunction with the LBM.

11.4.1 Introduction

Boundary conditions in the LBM are usually treated on the population level, i.e. the populations f_i are manipulated or constructed in such a way that the desired values for pressure and velocity (or their derivatives) are obtained at the boundary. This applies to all boundary conditions discussed in Chap. 5 and in the present chapter up to this point.

There is, however, a completely different way to enforce boundary conditions which was available long before anybody knew of the LBM. In 1972, Peskin proposed the *immersed boundary method* (IBM) in his dissertation [71], followed by an article in 1977 [72]. Peskin's idea was to use the force density $\boldsymbol{F}(\boldsymbol{x}, t)$ in the Navier-Stokes equation to mimic a boundary condition. To this end, $\boldsymbol{F}(\boldsymbol{x}, t)$ has to be computed such that the fluid behaves *as if* there was a boundary with desired properties (e.g. no-slip). When correctly applied, this approach can be used to recover immersed rigid or deformable objects with nearly arbitrary shape. Since the boundary condition exists only on the Navier-Stokes level (*via* the force density $\boldsymbol{F}(\boldsymbol{x}, t)$), IBM is not aware of the populations f_i.

The IBM provides a number of advantages. The main advantage is its front-tracking character, i.e. the shape of the boundary is directly known and does not have to be reconstructed (as in phase-field or level-set approaches). Neither do intersection points have to be computed (as required for nearly all boundary conditions presented in this chapter so far). The IBM can be combined with any Navier-Stokes solver which supports external forcing, such as the LBM. The IBM is relatively simple to implement and, if done so properly, its numerical overhead is small. Moving and deformable boundaries can be realised without remeshing. It has to be noted that fluid exists on both sides of an IBM surface. In particular, closed surfaces are filled with fluid.

The original IBM does not take any consideration of the kinetic origin of the LBM as it only operates on the Navier-Stokes level. Still, the combination of the IBM and the LBM, also called immersed-boundary-lattice-Boltzmann method (IB-LBM), first proposed by Feng and Michaelides [74], has become a popular application. It therefore deserves a somewhat thorough introduction in this book, together with some recent developments and related approaches. We cannot provide an exhaustive coverage of the IBM in general, though. Readers who are interested in the IBM independently of the LBM should read the seminal paper by Peskin [73] and the review by Mittal and Iaccarino [77].

There is some dissent in the literature what "immersed boundary method" actually means and how it is defined. Some people use it for nearly all methods where a boundary is immersed in a fluid, including, for example, fictitious domain methods (Sect. 11.3). Here, we define those methods as immersed boundary methods which involve, on the one hand, an Eulerian grid and Lagrangian markers and, on the other hand, some kind of velocity interpolation and force spreading as devised by Peskin [73], but there is no clear distinction between the IBM and related

methods. Another way of putting this is the following.[5] In IBM we have marker points without mass that move exactly with the fluid. Through some mechanical model (e.g. a constitutive model for a deformable membrane or a penalty force for a rigid body), we can compute forces at these points which we apply to the fluid directly, rather than to the mechanical model itself.

In the remainder of this section we will focus on the IBM combined with LBM using the BGK collision operator. However, several authors recently pointed out that the MRT or TRT collision operators can bring additional advantages by reducing undesired velocity slip at the immersed boundaries [78, 79]. We will not discuss those extensions further.

11.4.2 Mathematical Basis

We will now review the original IBM, discuss its mathematical properties and show its basic numerical algorithm.

11.4.2.1 Eulerian and Lagrangian Systems

Mathematically, the basis of the IBM is an *Eulerian* and a *Lagrangian* system. The former is represented by a fixed regular grid on which the fluid lives and the Navier-Stokes equations are solved. The latter is an ensemble of marker points $\{\boldsymbol{r}_j\}$. They can be (nearly) arbitrarily distributed in space, as long as they are sufficiently dense (see below). These markers represent discrete surface points of the boundary and are generally allowed to move: $\boldsymbol{r}_j = \boldsymbol{r}_j(t)$. We therefore have to distinguish between two node systems (Fig. 11.10) with the following properties:

1. The Eulerian grid defined by the LBM lattice nodes (coordinates designated by $\boldsymbol{x}$) is regular and stationary.
2. The immersed boundary marker points $\boldsymbol{r}_j(t)$ are Lagrangian nodes. They are not bound to the Eulerian grid and can move in space.

If the boundary is rigid, one would ideally fix relative distances such that $|\boldsymbol{r}_{jk}(t)| = |\boldsymbol{r}_j(t) - \boldsymbol{r}_k(t)| = \text{const}$. This is often not achievable, and a somewhat softened condition $|\boldsymbol{r}_{jk}(t)| \approx \text{const}$ is used instead. For deformable boundaries, a relative marker motion is actually desired.

It may or may not be necessary to connect neighbouring markers. Most implementations of rigid boundary conditions do not require connected markers, while all deformable algorithms require some kind of surface tessellation which involves defining the markers *and* their connectivity (surface mesh). This mesh is called *nonconforming* as it does not have to be aligned with the lattice of the LBM. The

[5] Thanks to Eric Lorenz for suggesting this description.

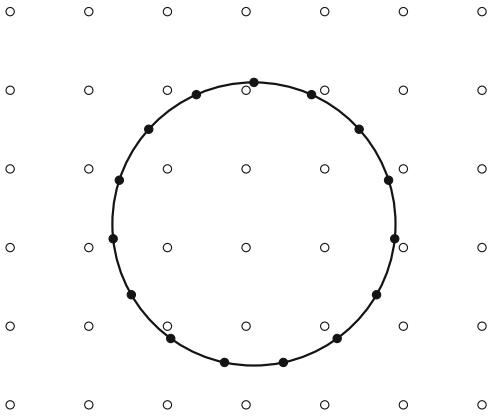

Fig. 11.10 Cylinder with boundary markers arbitrarily positioned in the regular fluid domain. The Eulerian mesh (fluid nodes, *open circles*) and the Lagrangian mesh (boundary, *solid nodes*) are independent. No intersections of lattice links with the boundary have to be computed

main advantage of the IBM is that the complex shape of the boundary is not related to the lattice structure and no intersection points have to be computed. This makes the IBM particularly useful for deformable boundaries (Sect. 11.4.5).

The decomposition of the geometry into two coordinate systems brings up the important question how to couple the dynamics of the boundary and the fluid. We need a bi-directional coupling where the fluid has to know about the presence of the boundary and *vice versa*. It is therefore required to communicate some information between both node systems through *velocity interpolation* and *force spreading*.

11.4.2.2 Continuous Governing Equations

We start with a fully continuous description and later turn our attention to the discretised version. In the following we assume the validity of the no-slip boundary condition, which is the first key idea of the IBM. It implies that each point of the surface $\boldsymbol{r}(t)$ and the ambient fluid at position $\boldsymbol{r}$ have to move with the same velocity:

$$\dot{\boldsymbol{r}}(t) = \boldsymbol{u}(\boldsymbol{r}(t), t). \tag{11.29}$$

The time dependence on the right-hand side of (11.29) is, on the one hand, caused by the variation of $\boldsymbol{u}$ itself and, on the other hand, by the boundary moving and therefore seeing different parts of the flow field. We can rewrite (11.29) as

$$\dot{\boldsymbol{r}}(t) = \int \mathrm{d}^3 x\, \boldsymbol{u}(\boldsymbol{x}, t)\delta(\boldsymbol{x} - \boldsymbol{r}(t)) \tag{11.30}$$

where $\delta(\boldsymbol{x} - \boldsymbol{r}(t))$ is Dirac's delta distribution. Equation (11.30) is the first of two governing equations of the as yet continuous IBM. We will later see that the discretised version of (11.30) requires *velocity interpolation*. Note that we write all

equations for 3D applications, but everything said (unless otherwise stated) can be directly applied in 2D as well.

The second governing equation describes the momentum exchange between the boundary and the fluid. In the IBM picture, we are interested in the force the boundary surface exerts on the nearby fluid, rather than the other way around. Let us assume we know the force density (per area) $\boldsymbol{F}_A(\boldsymbol{r}(t), t)$ everywhere on the boundary surface. Therefore, $\boldsymbol{F}_A \, \mathrm{d}^2 r$ is the force acting on a small area element $\mathrm{d}^2 r$.[6] The force density (per volume) that the fluid feels due to the presence of the immersed boundary can then be written as

$$\boldsymbol{F}(\boldsymbol{x}, t) = \int \mathrm{d}^2 r \, \boldsymbol{F}_A(\boldsymbol{r}(t), t) \delta(\boldsymbol{x} - \boldsymbol{r}(t)). \tag{11.31}$$

The delta distribution is the same as in (11.30). Equation (11.31) essentially means that the Lagrangian boundary force is spread to the Eulerian fluid. Therefore this equation is called *force spreading*.

Note that $\boldsymbol{F}(\boldsymbol{x}, t)$ is singular. Since the delta distribution $\delta(\boldsymbol{x} - \boldsymbol{r}(t))$ is 3D, but the integration is only 2D along the boundary, $\boldsymbol{F}(\boldsymbol{x}, t)$ is singular when crossing the boundary in normal direction. This marks the defining difference between velocity interpolation (which is non-singular) and force spreading.

Equation (11.30) and (11.31) are the basic IBM equations in their continuous form. We will now show their discretised versions which can be used in computer simulations.

11.4.2.3 Discretised Governing Equations

Since the velocity field is only known at discrete lattice sites, the integral in equation (11.30) cannot be exactly computed in a lattice-based simulation. The same holds for (11.31). Instead, both integrals have to be replaced by sums with a suitably chosen discretisation of the delta distribution.

Peskin [73] provided a full derivation of a general set of equations for the IBM. We will restrict ourselves, for the sake of brevity and clarity, to the final set of equations based on the assumption that the fluid in the entire volume is homogeneous, in particular its density and viscosity. We further assume that the markers are massless, which means that the boundary has the same density as the surrounding fluid.

[6] Remember that the boundary in 3D is a 2D surface.

The **discretised IBM equations** read

$$\dot{\boldsymbol{r}}_j(t) = \sum_{\boldsymbol{x}} \Delta x^3 \boldsymbol{u}(\boldsymbol{x},t)\Delta(\boldsymbol{r}_j(t),\boldsymbol{x}) \tag{11.32}$$

and

$$\boldsymbol{F}(\boldsymbol{x},t) = \sum_j \boldsymbol{f}_j(t)\Delta(\boldsymbol{r}_j(t),\boldsymbol{x}). \tag{11.33}$$

The fluid is discretised as an Eulerian lattice with coordinates $\boldsymbol{x}$, the boundary is approximated by an ensemble of markers at $\boldsymbol{r}_j(t)$. Here, $\boldsymbol{u}$ and $\boldsymbol{F}$ are velocity and force density (per volume) on the lattice, $\dot{\boldsymbol{r}}_j$ and $\boldsymbol{f}_j$ are the velocity of and the total force acting on the markers. Velocity interpolation in (11.32) and force spreading in (11.33) are the central IBM equations. Both require an appropriate kernel function (or stencil) Δ, as discussed below.

It is important to realise that $\boldsymbol{f}_j(t)$ is the total force (not force density) acting on node j at position $\boldsymbol{x}_j(t)$. Apart from the no-slip condition discussed above, it is one of the key ideas of the IBM that the force $\boldsymbol{f}_j(t)$ is first computed in the Lagrangian system and then spread to the lattice. This brings up the central question how $\boldsymbol{f}_j(t)$ can be found in the first place. In fact, this depends strongly on the chosen kind of the IBM. We will discuss this problem in the upcoming sections. Let us for now simply assume that all forces $\boldsymbol{f}_j(t)$ are known at each time step.

We further emphasise that $\boldsymbol{F}(\boldsymbol{x},t)$ is the only mechanism through which the fluid is aware of the presence of the boundary; there is otherwise no direct boundary condition for the fluid. Once we know $\boldsymbol{F}(\boldsymbol{x},t)$, we can use one of the forcing schemes described in Chap. 6 to update the fluid. To the LBM, the IBM force density is not at all different from gravity (although gravity is usually homogeneous and constant).

11.4.2.4 Kernel Functions

The function $\Delta(\boldsymbol{r}_j,\boldsymbol{x})$ is a suitably discretised version of the Dirac delta distribution and another key ingredient of any IBM. It is in most cases simplified by assuming $\Delta(\boldsymbol{r}_j,\boldsymbol{x}) = \Delta(\boldsymbol{r}_j-\boldsymbol{x})$, i.e. it is only a function of the distance vector $\boldsymbol{r}_j-\boldsymbol{x}$ rather than a more general function of $\boldsymbol{r}_j$ and $\boldsymbol{x}$ (see [80] for a kernel without this simplification). It is not directly obvious which functions $\Delta(\boldsymbol{r}_j-\boldsymbol{x})$ qualify as valid interpolation and spreading kernels. While Peskin [73] explains the procedure to find suitable discretisations in detail and an overview of interpolation function can be found in [76, 81], we will only provide the basic ideas and final results.

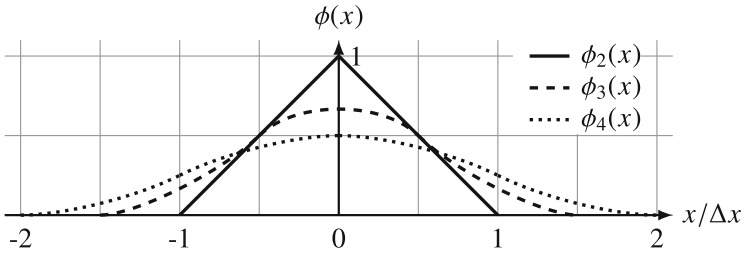

Fig. 11.11 IBM interpolation stencils ϕ_2, ϕ_3, and ϕ_4. The total kernel range is two, three and four lattice sites, respectively

The fundamental claims and restrictions are:

- Interpolation and spreading should be short-ranged. This is required to reduce computational overhead by making the number of summands in (11.32) and (11.33) as small as possible.
- Momentum and angular momentum have to be identical when evaluated either in the Eulerian or the Lagrangian system (same speed and rotation in both systems).
- Lattice artefacts ("bumpiness" of the interpolation when boundaries move) should be suppressed as much as possible.
- The kernel has to be normalised: $\sum_{\boldsymbol{x}} \Delta x^3 \Delta(\boldsymbol{x}) = 1$.

It is convenient to factorise the kernel function as $\Delta(\boldsymbol{x}) = \phi(x)\phi(y)/\Delta x^2$ in 2D and $\Delta(\boldsymbol{x}) = \phi(x)\phi(y)\phi(z)/\Delta x^3$ in 3D, i.e. each major coordinate axis contributes independently. This is not essential but simplifies the procedure. Peskin [73] derived a series of stencils which are also shown in Fig. 11.11. Those kernels read

$$\phi_2(x) = \begin{cases} 1 - |x| & (0 \le |x| \le \Delta x) \\ 0 & (\Delta x \le |x|) \end{cases}, \tag{11.34}$$

$$\phi_3(x) = \begin{cases} \frac{1}{3}\left(1 + \sqrt{1 - 3x^2}\right) & 0 \le |x| \le \frac{1}{2}\Delta x \\ \frac{1}{6}\left(5 - 3|x| - \sqrt{-2 + 6|x| - 3x^2}\right) & \frac{1}{2}\Delta x \le |x| \le \frac{3}{2}\Delta x \\ 0 & \frac{3}{2}\Delta x \le |x| \end{cases}, \tag{11.35}$$

$$\phi_4(x) = \begin{cases} \frac{1}{8}\left(3 - 2|x| + \sqrt{1 + 4|x| - 4x^2}\right) & 0 \le |x| \le \Delta x \\ \frac{1}{8}\left(5 - 2|x| - \sqrt{-7 + 12|x| - 4x^2}\right) & \Delta x \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}. \tag{11.36}$$

The integer index denotes the number of lattice nodes required for interpolation and spreading along each coordinate axis (Fig. 11.11). Therefore, the stencils require 2^d, 3^d and 4^d lattice sites in d dimensions, respectively.

ϕ_4 fulfills all of Peskin's requirements, but it also leads to a diffuse boundary since the interpolation range is rather large. ϕ_3 also fulfills all requirements, but it is less smooth. ϕ_2 is most efficient in terms of computing time and leads to the sharpest boundaries, but the lattice structure is not well hidden, i.e. the resulting flow field is generally more bumpy. Sometimes, $\phi_4(x)$ is replaced by another stencil which has nearly the same shape but does not exactly satisfy all of the requirements mentioned above:

$$\phi_4'(x) = \begin{cases} \frac{1}{4}\left(1 + \cos(\frac{\pi x}{2})\right) & 0 \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}. \tag{11.37}$$

We also note that additional, smoothed representations of the delta distribution have been proposed [82].

11.4.2.5 General IB-LBM Algorithm

Without specifying yet how the forces $\boldsymbol{f}_j$ acting on the boundary nodes are obtained, we can still jot down a simple IB-LBM algorithm. It consists of the following substeps:

1. Compute the Lagrangian forces $\boldsymbol{f}_j(t)$ from the current boundary configuration $\{\boldsymbol{r}_j(t)\}$. This is a model-dependent step which still remains to be discussed.
2. Spread the Lagrangian forces $\boldsymbol{f}_j(t)$ to the lattice *via* (11.33) to obtain the Eulerian force density $\boldsymbol{F}(\boldsymbol{x}, t)$. See also Fig. 11.12.
3. Compute the uncorrected (pre-collision) velocity $\boldsymbol{u}(\boldsymbol{x}, t)$ from $\boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i / \rho$.
4. Perform the LB algorithm (computing equilibrium distributions, collision and propagation) with forcing (Chap. 6), using $\boldsymbol{u}(\boldsymbol{x}, t)$ and $\boldsymbol{F}(\boldsymbol{x}, t)$ as input. If other forces, such as gravity, are present, the total force is the sum of all these contributions. Note that the choice of an accurate forcing scheme (e.g. Guo et al. [83]) is important. This is often ignored in the literature.

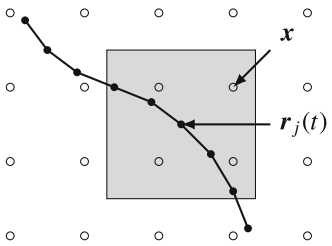

Fig. 11.12 Interpolation and spreading. The lattice velocity is interpolated at $\boldsymbol{r}_j(t)$. For this operation, all lattice nodes within the grey region are required (here ϕ_2 is used). The force density at a given lattice node $\boldsymbol{x}$ is the sum of all contributions from those nodes $\boldsymbol{r}_j(t)$ whose interpolation box covers $\boldsymbol{x}$

5. As we know from Chap. 6, the physical fluid velocity during the time step is given by the first moment of the populations and a force correction:

$$\boldsymbol{u}_{\mathrm{f}}(\boldsymbol{x},t) = \boldsymbol{u}(\boldsymbol{x},t) + \frac{\boldsymbol{F}(\boldsymbol{x},t)\Delta t}{2\rho(\boldsymbol{x},t)}. \tag{11.38}$$

   Leaving the force correction out can lead to significant stability (and accuracy) problems.
6. Interpolate the fluid velocity $\boldsymbol{u}_{\mathrm{f}}(\boldsymbol{x},t)$ at the Lagrangian node positions *via* (11.32) to obtain $\dot{\boldsymbol{r}}_j(t)$. See also Fig. 11.12.
7. Advect the boundary nodes (usually by the explicit forward Euler method) to find the new boundary configuration:

$$\boldsymbol{r}_j(t+\Delta t) = \boldsymbol{r}_j(t) + \dot{\boldsymbol{r}}_j(t)\Delta t. \tag{11.39}$$

   There exist different explicit time integration schemes [84, 85], though.
8. Go back to step 1 for the next time step.

Note that the time steps for the LBM and the marker position update are identical, i.e. in the standard IB-LBM there can only be one marker update per LB time step.

Not all IBM flavours follow this algorithm. There are several approaches (and algorithms) to deal with rigid boundaries. We will get back to those later.

Once the discretised kernel functions $\Delta(\boldsymbol{x})$ have been implemented, the rules for computing the forces $\boldsymbol{f}_j$ have been defined and the initial boundary node locations $\boldsymbol{r}_j(t=0)$ are known, the simulation can be executed. The real challenge is normally hidden behind the models providing the required forces $\boldsymbol{f}_j$. We will get back to this point in the following sections.

For the sake of **efficiency**, note that is it very easy to implement a naive IBM which is, despite being mathematically correct, horribly inefficient. Equation (11.32) clearly shows that the sum should run over the lattice neighbours of a given boundary node. For a boundary node $\boldsymbol{r}_j$, it is easy to identify the neighbouring lattice sites. However, (11.33) suggests to go the other way around and to identify all boundary markers in the vicinity of a given lattice site. This can be extremely expensive, in particular when the Eulerian lattice is large. A small trick can make the computational effort for interpolation and spreading identical though. In order to do so, we run over all known boundary markers $\boldsymbol{r}_j$ and compute the fraction of the force density a neighbouring lattice site $\boldsymbol{x}$ would receive:

$$\delta_j \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{f}_j(t)\Delta(\boldsymbol{r}_j(t),\boldsymbol{x}). \tag{11.40}$$

(continued)

Here, $\delta_j \boldsymbol{F}(\boldsymbol{x})$ is the contribution to $\boldsymbol{F}(\boldsymbol{x})$ due to the presence of $\boldsymbol{r}_j$ alone. All these contributions are simply summed and the correct total force density $\boldsymbol{F}(\boldsymbol{x}) = \sum_j \delta_j \boldsymbol{F}(\boldsymbol{x})$ is automatically obtained in the end. This also highlights the conceptual difference between interpolation and spreading. In fact, Tryggvason et al. [86] give the helpful advice (where "front" means "boundary" in our case):

> When information is transferred between the front and the fixed grid, it is always easier to go from the front to the grid and not the other way around. Since the fixed grid is structured and regular, it is very simple to determine the point on the fixed grid that is closest to a given front position.

11.4.2.6 Implications of the Combination of IBM and LBM

Although the IBM is just another way to impose boundary conditions on the Navier-Stokes level, the populations f_i are completely unimpressed by the presence of the Lagrangian marker points. In particular, the f_i simply penetrate any closed IB surface. This is not problematic as long as one is only interested in the no-slip condition of the velocity $\boldsymbol{u}$ and one does not care what the populations are doing. But we can already see that the IBM is not an ideal approach when one wants to keep, for example, two fluids separate on two sides of a membrane.

Another observation is that the fluid usually fills the entire space, including the regions inside any boundary. This significantly simplifies things but can also lead to additional difficulties. For example, it has been shown that the dynamics of the interior fluid can have an effect on the dynamics of the exterior fluid if the immersed boundaries are rotating [87]. There are ways around this, for example by adding *interior* marker points. In the following, we will not discuss methods with interior markers, such as direct-forcing/fictitious-domain methods [88].

11.4.2.7 Distribution of Markers in Space

One open question is how to distribute the markers $\boldsymbol{r}_j$ in space initially. For 2D problems this answer is easy to answer: define a 1D chain of markers with a given mutual distance d on the boundary. Each marker knows which one is its left and right neighbour.

The choice of d is a more delicate issue. On the one hand, intuitively, d cannot be too large because otherwise there are "holes" in the boundary and fluid can flow between markers. On the other hand, too small a value for d can lead to problems as well [89]. This is due to the peculiarities of the IBM algorithm: the marker position update relies on the interpolated fluid velocity. If two markers are very close, $d \ll \Delta x$, they essentially see the same fluid environment and move with the

same velocity. Markers which are too close can therefore not be separated again (or only with a lot of effort) and they can stick together. It is usually recommended to choose d somewhere between 0.5 and one lattice constant, but some authors even choose $d \approx 2\Delta x$. We will get back to this point in Sect. 11.4.3.

The situation is much more complicated in 3D where boundaries are generally curved 2D surfaces. One has to distribute the markers such that the mutual distance of any pair of neighbours is approximately the same. This can be a tedious task for general surface shapes and is one of the biggest challenges when applying the IBM in 3D. Furthermore, the node connectivity (i.e. an unstructured mesh) is required for deformable boundaries and additional constraints may apply in those situations (e.g. the resulting triangular face elements should be as equilateral as possible). Here, we can only give some starting points for further literature studies:

- For simple geometries of high symmetry (spheres, red blood cells), one can start from an *icosahedron* and subdivide each triangular surface element into n^2 ($n > 1$ being an integer) triangular elements [85, 90]. The markers are radially or tangentially shifted to approximate the desired boundary shape.
- In the *minimum potential approach* [91] a fixed number of markers is initially randomly distributed on the surface. Markers interact via repulsive forces and move along the surface until the system has found an energetic minimum. The resulting marker configuration can then be used in simulations as initial boundary discretisation.
- Feng and Michaelides [91] presented another approach to distribute markers on a sphere by defining parallel segments containing equidistant nodes.
- There exist free meshing tools which can cope with more complicated boundary shapes, for example [92, 93].

11.4.2.8 Accuracy and Convergence

One shortcoming of the IBM is that the velocity interpolation does not generally maintain the solenoidal properties of the fluid. Even if the fluid solver is perfectly divergence-free (which LBM usually cannot claim), the interpolated velocity may not be divergence-free. The consequence is that the volume of an enclosed region can change in time.

Furthermore, the IBM is formally a first-order accurate boundary condition [73]. There seems to be some dispute in the literature about the actual convergence rate, though. While Peng and Luo [94] report second-order convergence, other authors observed only first-order convergence for the velocity field [95, 96].

Related to the question of accuracy and convergence is the apparent size of particles and/or apparent location of walls modelled with the IBM. Several authors, e.g. [85, 91, 97], have reported that particles appear to be larger than they actually are. Instead of the input radius r, a larger radius $r + \delta r$ is observed where δr is somewhere between $0.2\Delta x$ and $0.5\Delta x$, depending on the chosen stencil (a kernel with wider support usually leads to a larger δr). In order to model a sphere of actual

radius r, Feng and Michaelides [91] suggested to distribute markers on a sphere with radius

$$r_\mathrm{b} = \sqrt[3]{\frac{r^3 + (r - \Delta x)^3}{2}} \tag{11.41}$$

instead. This finding is important for the modelling of particle suspensions or porous media where the rheology strongly depends on the volume fraction and porosity. We will get back to the convergence and apparent wall location in Sect. 11.4.3.

Once again, we see that there is no free lunch. The advantages of the standard IBM (ease of implementation, no need to find boundary intersections, no treatment of fresh fluid nodes required), which explain the IBM's popularity, are challenged by inferior accuracy and convergence compared to other boundary conditions. Note, however, that there have been efforts to make the IB-LBM more accurate [78, 79, 97].

11.4.3 Explicit Feedback IBM for Rigid Boundaries

We show a simple way to compute the nodal forces $\boldsymbol{f}_j$ for (nearly) rigid boundaries. This *explicit* IBM is easy to implement but shows weak stability properties. After discussing the algorithm we use the explicit IBM to model Poiseuille flow and demonstrate the convergence and boundary location issues within the IBM. Note that the explicit IBM does not work very well for unsteady flows as it takes some time for the marker points to respond to the flow.

11.4.3.1 Algorithm

A rigid body is defined by $|\boldsymbol{r}_j(t) - \boldsymbol{r}_k(t)| = \text{const}$ for any two points $\boldsymbol{r}_j$ and $\boldsymbol{r}_k$ of the body. The simplest way to approximate rigid objects with the IBM is to model the boundary as a collection of marker points $\boldsymbol{r}_j(t)$ which are individually connected by an elastic spring to their reference locations $\boldsymbol{r}_j^{(0)}(t)$. Feng and Michaelides [74] first proposed this idea within the framework of IB-LBM in 2004. While the virtual reference locations obey the rigidity condition exactly, $|\boldsymbol{r}_j^{(0)}(t) - \boldsymbol{r}_k^{(0)}(t)| = \text{const}$, the real markers are allowed to deviate slightly from this condition.

The magnitude of the undesired body deformation can be controlled by springs with strength κ. We can then explicitly compute the marker "penalty" force $\boldsymbol{f}_j$ from a function like

$$\boldsymbol{f}_j(t) = -\kappa \delta \boldsymbol{r}_j(t), \quad \delta \boldsymbol{r}_j(t) = \boldsymbol{r}_j(t) - \boldsymbol{r}_j^{(0)}(t) \tag{11.42}$$

at each time step so that the required nodal forces $\boldsymbol{f}_j$ are known. Contrarily, Feng and Michaelides [74] proposed a form similar to

$$\boldsymbol{f}_j(t) = \begin{cases} 0 & |\delta \boldsymbol{r}_j(t)| = 0 \\ -\kappa \frac{\delta \boldsymbol{r}_j(t)}{|\delta \boldsymbol{r}_j(t)|} & |\delta \boldsymbol{r}_j(t)| > 0 \end{cases}. \tag{11.43}$$

In the example shown below, we use another penalty force:

$$\boldsymbol{f}_j(t) = -\kappa \frac{d}{\Delta x} \delta \boldsymbol{r}_j(t). \tag{11.44}$$

The difference to (11.42) is that the *force per node* is weighted by the average distance d between the nodes. This guarantees that increasing the number of markers (and therefore decreasing d) does not increase the *total force* at the boundary. In any case, the IBM algorithm in Sect. 11.4.1 is employed: in step 1, the forces $\boldsymbol{f}_j(t)$ are obtained *via* one of the approaches shown above.

Using the explicit penalty IBM, each marker point is allowed to be slightly carried away from its reference position. Each point applies a penalty force as discussed above. This force then tends to pull the marker back towards its reference position. After a few time steps (given a steady flow), a marker point will reach an "equilibrium position" where the force it exerts on the fluid is just enough to keep the fluid, and therefore itself, in place. It has then achieved a no-slip condition locally.

Ideally, the exact form of the penalty force should not be important, but it depends on the chosen parameter values whether this is actually the case. For example, if κ is too small, the undesired deformation becomes too large, and if κ is too large, the simulation can become unstable. A clear disadvantage of this method is that the optimum range for κ has to be obtained and that a small time step may be necessary. It is not possible to achieve perfectly rigid boundaries with an explicit IBM algorithm.

Finally, we distinguish between three fundamental cases:

1. The rigid body is fixed in space. All reference points $\boldsymbol{r}_j^{(0)}$ are stationary, and their positions do not have to be updated. The Poiseuille flow in the example below belongs to this category.
2. The body is rigid, and its motion is externally prescribed. This is similar to the first case, but the marker point positions $\boldsymbol{r}_j^{(0)}$ are updated according to the a priori known velocity.
3. The body is rigid but can move *freely* in space. This means that the reference points $\boldsymbol{r}_j^{(0)}$ have to be updated according to the equations of motion of a rigid body. In contrast to the second case, this requires the momentum and angular momentum exchange to be integrated on the surface of the body to find the total force and torque acting on the body. Updating the marker positions of rigid bodies can be complicated. We will not discuss details here and instead refer to the literature [74, 87, 91, 98].

11.4.3.2 Stationary Boundary: Poiseuille Flow

We simulate a force-driven Poiseuille flow along the x-axis in 2D with (nearly) rigid boundaries as shown in Fig. 11.13. The gravitational force density driving the flow is $F = 10^{-5}$, and the fluid domain consists of $N_x \times N_y = 19 \times 20$ nodes on a D2Q9 lattice. The BGK collision operator with $\tau = \Delta t$ is used. We approximate both walls by lines of markers with mutual distance d as free parameter and employ the penalty force in (11.44). The distance between the IBM walls is $D = 15.3\Delta x$. The spring constant κ is the second free parameter. Simulations are run until the velocity profile is stationary.

We have chosen a prime number for N_x and a non-integer for D to reduce the symmetry of the problem and therefore avoid situations which may accidentally have small numerical errors. Note, however, that the chosen benchmark problem is still highly idealised. Typical IBM applications involve moving curved boundaries with complex shapes. The purpose of this exercise is to get an initial feeling for the IBM simulation parameters.

The first task is to investigate the effect of the Lagrangian mesh spacing $d/\Delta x$. We keep $\kappa = \Delta t$ fixed and vary d for two interpolations stencils, ϕ_2 and ϕ_4. As error measure we take the largest value of u_y in the simulation, normalised by the Poiseuille peak velocity $\hat{u}_x$. Note that ideally we expect $u_y = 0$ everywhere. The results are shown in Fig. 11.14a. The ϕ_2-errors are larger than the ϕ_4-errors for $d > \Delta x$, but they are smaller for $d < \Delta x$. A resonance effect with vanishing u_y can be seen for $d = \Delta x$ and $d = 0.5\Delta x$. In those situations the problem is highly symmetric as the system is x-periodic after a single lattice unit. Generally we conclude that d should not be larger than $1.5\Delta x$. Cheng et al. [76] reported a similar

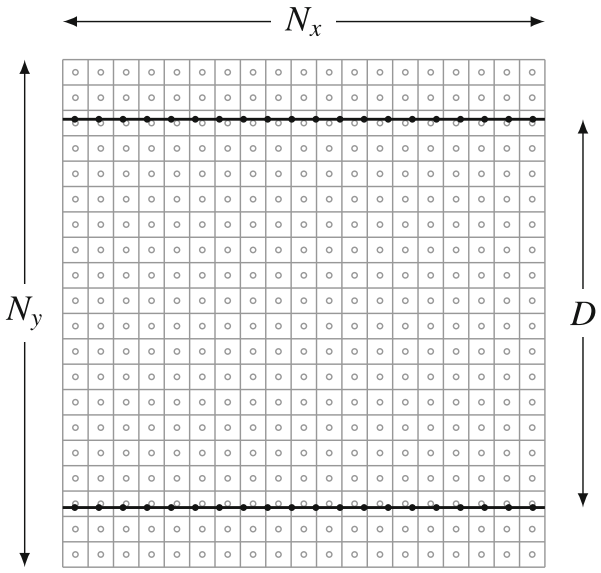

Fig. 11.13 Setup of the Poiseuille flow problem. The lattice size is $N_x \times N_y = 19\Delta x \times 20\Delta x$, and the distance between the IBM walls is $D = 15.3\Delta x$. In this particular example, $d = 0.95\Delta x$ is chosen

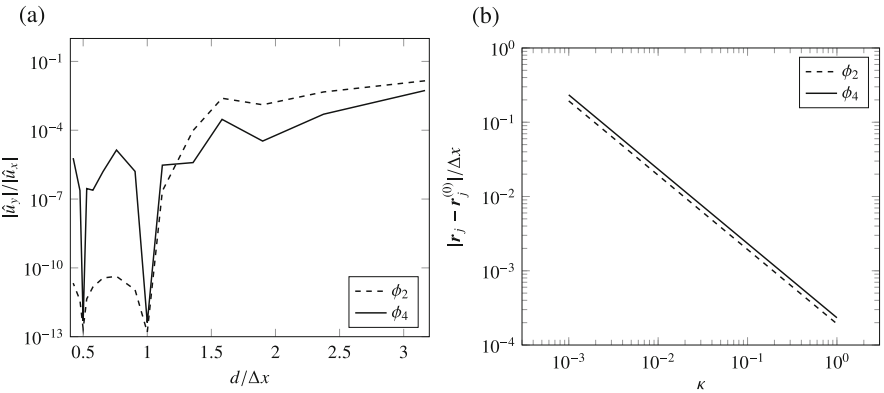

Fig. 11.14 Benchmark results showing the effect of mesh spacing d and penalty parameter κ on the accuracy of the explicit IBM for Poiseuille flow. (**a**) Sensitivity to mesh spacing. (**b**) Sensitivity to penalty parameter

observation. For this simple example, ϕ_2 provides significantly better results than ϕ_4, but this observation should certainly not be generalised to arbitrary situations. The resonance effect is expected to disappear for more complex geometries with curved boundaries.

In the second test, we set $d = \Delta x$ and vary the penalty parameter κ. Due to the explicitness of the algorithm, the Lagrangian nodes are slightly dragged by the fluid along the x-axis until the penalty force balances the drag force. We show the displacement of the Lagrangian nodes as function of penalty parameter κ in Fig. 11.14b. As expected, the displacement is inversely proportional to the penalty parameter. For $\kappa = 1$, the displacement is less than 0.1% of a lattice spacing Δx, which should be sufficient for most applications. We found that $\kappa > 3$ leads to instability. Concluding, $d = \Delta x$ and $\kappa = 1$ are reasonable choices for the current problem; we will keep these values for the final tests. Note, however, that different flow configurations may require different parameter values for optimum results.

We now investigate the apparent boundary location and the convergence rate of the IBM. For that purpose, we perform a grid refinement study. We only vary the system size, but keep $d = \Delta x$, $\kappa = 1$ and $\tau = \Delta t$ fixed (diffusive scaling). As a consequence, the gravitational force density F scales with $(D/\Delta x)^{-3}$ and the expected peak velocity $\hat{u}_x$ with $(D/\Delta x)^{-1}$ (cf. Chap. 7). For each simulation, we fit a parabola to the flow field in the central region between $\pm D/2$ and compute the appararent channel diameter D_{app}. Figure 11.15a shows the mismatch of the channel diameter, $D_{\text{app}} - D$, as function of resolution. Obviously the channel appears to be smaller than expected, which also leads to a reduced peak velocity compared to its expected values (not shown here). The mismatch is larger for ϕ_4 than for ϕ_2. Futhermore, the diameter mismatch does not significantly depend on the resolution. This means that the mismatch cannot be removed by increasing the resolution, which leads only to a first-order convergence rate of the velocity error.

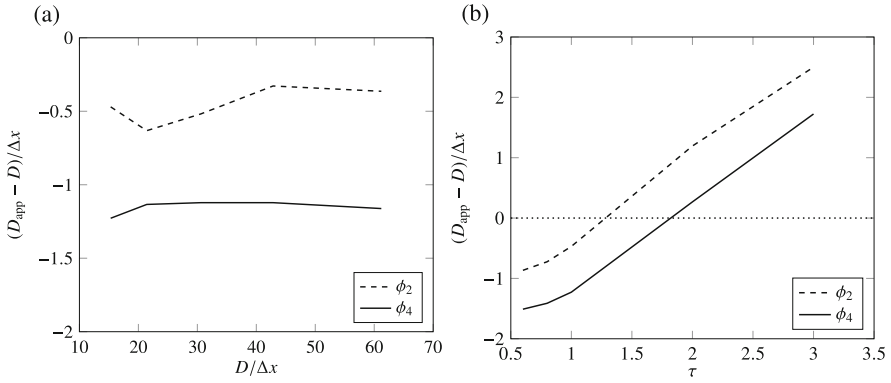

Fig. 11.15 Channel diameter mismatch as function of spatial resolution and LBM relaxation time τ. (**a**) Sensitivity to resolution. (**b**) Sensitivity to relaxation time

In the final test we investigate how the wall location mismatch depends on the relaxation time τ. We now vary τ at fixed resolution. Figure 11.15b reveals that D_{app} is a function of τ. Depending on the value of τ, the channel can appear smaller or larger than expected. While the exact values depend on the choice of the interpolation stencil, we can conclude that the appararent channel diameter increases roughly linearly with τ for $\tau > \Delta t$. This is a highly undesirable effect that has been discussed by several authors [76, 79, 99]. It has recently been suggested to use the MRT [78] or TRT [79] collision operators to resolve this problem. We will not discuss these approaches here.

As already concluded at the end of Sect. 11.4.2, the IBM accuracy is typically inferior to other available boundary conditions. Care has to be taken when the exact channel diameter or particle size (for suspension simulations) is important. In the end, it can take a significant amount of work to make sure that an IBM code is working reliably.

11.4.4 Direct-Forcing IB-LBM for Rigid Boundaries

The explicit penalty IBM for rigid boundaries has a major disadvantage: it involves a free parameter whose choice affects the stability and accuracy. We seek an alternative implementation without a free parameter. This means that the IB force has to be computed directly from the flow field. Therefore, we call this class of methods *direct-forcing IB-LBM*.

Feng and Michaelides [91, 100] originally combined the direct forcing IBM with the LBM. A number of alternative direct-forcing IB-LBMs have been proposed

since then. We can distinguish between three different approaches, each with different levels of accuracy and numerical cost:

1. implicit IBM
2. multi-direct-forcing IBM (iterative)
3. direct-forcing IBM (explicit)

We cannot present and compare all available approaches in depth. Instead, we will start from the underlying hydrodynamic problem and show which steps are necessary to construct a reliable parameter-free IB-LBM.

11.4.4.1 Background

We assume a rigid boundary described by a number of marker points at positions $\boldsymbol{r}_j$. The markers have known velocities $\dot{\boldsymbol{r}}_j = \boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$, the desired boundary velocity. In most situations, the boundary is resting, but there is no fundamental difficulty with moving (translating and rotating) boundaries.

Before collision, the fluid velocity on a lattice node $\boldsymbol{x}$ is

$$\boldsymbol{u}(\boldsymbol{x}) = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i(\boldsymbol{x}) \boldsymbol{c}_i. \tag{11.45}$$

In the absence of a force, collision leaves the momentum and velocity invariant. The only mechanism that can change the fluid velocity during collision is a body force $\boldsymbol{F}$:

$$\boldsymbol{u}^\star(\boldsymbol{x}) = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i^\star(\boldsymbol{x}) \boldsymbol{c}_i = \frac{1}{\rho(\boldsymbol{x})} \sum_i f_i(\boldsymbol{x}) \boldsymbol{c}_i + \frac{\boldsymbol{F}(\boldsymbol{x}) \Delta t}{\rho(\boldsymbol{x})}. \tag{11.46}$$

As usual, a star denotes post-collision quantities. Furthermore, we drop the time t because everything which follows is happening within a single time step.

We know that the physical fluid velocity during a time step is the average of the pre- and post-collision velocities [83]:

$$\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}) = \frac{\boldsymbol{u}(\boldsymbol{x}) + \boldsymbol{u}^\star(\boldsymbol{x})}{2} = \boldsymbol{u}(\boldsymbol{x}) + \frac{\boldsymbol{F}(\boldsymbol{x}) \Delta t}{2\rho(\boldsymbol{x})}. \tag{11.47}$$

The central idea of any direct-forcing IB-LBM is to construct the force $\boldsymbol{F}(\boldsymbol{x})$ in such a way that $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ matches the known boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ at the marker positions to satisfy the no-slip condition. As we will now see, this is a non-trivial task and explains why there is a number of direct-forcing variants in the literature.

Since we are working in the framework of the IBM, the boundary velocity is known at the positions of the boundary markers: $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \dot{\boldsymbol{r}}_j$. This means that we have to interpolate $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ at the boundary markers $\boldsymbol{r}_j$ to obtain $\boldsymbol{u}_\mathrm{f}(\boldsymbol{r}_j)$ first, then find

the required boundary force $\boldsymbol{f}_j$ and finally spread the force back to the lattice to obtain $\boldsymbol{F}(\boldsymbol{x})$.

The difficulty is caused be the non-local velocity interpolation and force spreading. In order to compute $\boldsymbol{f}_j$, we require the velocity on all lattice nodes $\boldsymbol{x}$ close to $\boldsymbol{r}_j$. In return, we have to spread $\boldsymbol{f}_j$ back to those lattice nodes. However, many lattice nodes $\boldsymbol{x}$ participate in interpolation and spreading of more than one marker $\boldsymbol{r}_j$ *at the same time*. This means that (11.47) has to be solved *simultaneously* on all lattice nodes to guarantee a consistent solution.

11.4.4.2 Implicit IB-LBM

In 2009, Wu and Shu [101] proposed the *implicit velocity correction-based IB-LBM* which is probably the most accurate and consistent way to enforce the no-slip condition at a rigid boundary with the IB-LBM. We will only provide the derivation of the algorithm. Benchmark tests can be found in [101–103].

The basic idea of the implicit IB-LBM is to consider the required force density $\boldsymbol{F}(\boldsymbol{x})$ as the unknowns for which the problem has to be solved in such a way that the no-slip condition is satisfied. Since the unknowns $\boldsymbol{F}(\boldsymbol{x})$ depend on the current and the desired flow field, the problem is implicit.

The first step is to write the physical fluid velocity in (11.47) as

$$\boldsymbol{u}_\mathrm{f}(\boldsymbol{x}) = \boldsymbol{u}(\boldsymbol{x}) + \delta\boldsymbol{u}(\boldsymbol{x}) \tag{11.48}$$

where $\boldsymbol{u}(\boldsymbol{x})$, given by (11.45), is the known uncorrected velocity and $\delta\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{F}(\boldsymbol{x})\Delta t)/(2\rho(\boldsymbol{x}))$ is the unknown velocity correction required to achieve the desired no-slip condition.

Now, we use the IBM relations in (11.32) and (11.33) to link Eulerian and Lagrangian quantities. We can express the Eulerian correction terms $\delta\boldsymbol{u}(\boldsymbol{x})$ by their Lagrangian counterparts $\delta\boldsymbol{u}(\boldsymbol{r}_j)$:

$$\delta\boldsymbol{u}(\boldsymbol{x}) = \sum_j \delta\boldsymbol{u}(\boldsymbol{r}_j)\Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.49}$$

Note that $\delta\boldsymbol{u}(\boldsymbol{r}_j)$ is proportional to the unknown Lagrangian force $\boldsymbol{f}_j$.

Let $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ be the desired boundary velocity imposed on the Lagrangian nodes. The aim is construct an Eulerian flow field $\boldsymbol{u}_\mathrm{f}(\boldsymbol{x})$ which, interpolated at the boundary nodes, equals the boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$:

$$\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \sum_{\boldsymbol{x}} \boldsymbol{u}_\mathrm{f}(\boldsymbol{x})\Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.50}$$

To achieve this, we combine (11.48), (11.49) and (11.50):

$$\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \sum_{\boldsymbol{x}} \left[\boldsymbol{u}(\boldsymbol{x}) + \sum_k \delta\boldsymbol{u}(\boldsymbol{r}_k) \Delta(\boldsymbol{r}_k, \boldsymbol{x}) \right] \Delta(\boldsymbol{r}_j, \boldsymbol{x}). \tag{11.51}$$

The only unknowns in this equation are the desired correction terms $\delta\boldsymbol{u}(\boldsymbol{r}_k)$. We can rewrite this equation in the following way:

$$\sum_k \underbrace{\left[\sum_{\boldsymbol{x}} \Delta(\boldsymbol{r}_k, \boldsymbol{x}) \Delta(\boldsymbol{r}_j, \boldsymbol{x}) \right]}_{A_{jk}} \underbrace{\delta\boldsymbol{u}(\boldsymbol{r}_k)}_{X_k} = \underbrace{\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) - \sum_{\boldsymbol{x}} \boldsymbol{u}(\boldsymbol{x}) \Delta(\boldsymbol{r}_j, \boldsymbol{x})}_{B_j} \tag{11.52}$$

or, in simple matrix-vector notation, as $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$.

The vectors $\boldsymbol{X}$ and $\boldsymbol{B}$ have N elements, and $\boldsymbol{A}$ is an $N \times N$-matrix where N is the number of marker points, i.e. j and k run from 1 to N. The elements of $\boldsymbol{A}$ are functions of the node positions $\boldsymbol{r}_j$ only, depending on the choice of the IBM stencil Δ. Finding the unknowns $\boldsymbol{X}$ *via* $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$ requires inversion of $\boldsymbol{A}$. Obviously the matrix A can be large, with N typically ranging from several 10^2 to several 10^4 or 10^5.

Concluding, the implicit IB-LBM algorithm works as follows:

1. Compute the matrix $\boldsymbol{A}$ and its inverse $\boldsymbol{A}^{-1}$ from the known node positions $\boldsymbol{r}_j$. See [101] for details.
2. Stream the populations to obtain $f_i(\boldsymbol{x})$ and compute the density and uncorrected velocity from $\rho\boldsymbol{u} = \sum_i f_i \boldsymbol{c}_i$.
3. Using the known boundary velocity $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$ and the uncorrected fluid velocity $\boldsymbol{u}(\boldsymbol{x})$, solve the matrix equation, (11.52), for the unknown corrections $\delta\boldsymbol{u}(\boldsymbol{r}_j)$.
4. Spread $\delta\boldsymbol{u}(\boldsymbol{r}_j)$ to the Eulerian grid *via* (11.49).
5. Compute the desired force density from $\delta\boldsymbol{u}(\boldsymbol{x}) = (\boldsymbol{F}(\boldsymbol{x})\Delta t)/(2\rho(\boldsymbol{x}))$.
6. Perform collision with forcing.
7. If the boundary is stationary, i.e. all boundary velocities obey $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) = \boldsymbol{0}$, go back to step 2 for the next time step.
8. If the boundary is not stationary, update the positions $\boldsymbol{r}_j$ and velocities $\dot{\boldsymbol{r}}_j = \boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$. The position update may be enforced (e.g. oscillating cylinder) or a consequence of fluid stresses (e.g. freely moving cylinder). In the latter case, a suitable time integrator has to be chosen [74, 87, 91, 98].
9. Go back to step 1 for the next time step.

Note that the re-computation of the matrix $\boldsymbol{A}$ and its inverse at each time step for non-stationary boundaries can be expensive when N is large. Therfore, alternative approaches, such as multi-direct forcing, which are computationally more efficient and conceptually simpler, have been suggested.

11.4.4.3 Multi Direct-Forcing IB-LBM

The aim of *multi direct-forcing IB-LBM* is to avoid the construction and inversion of the matrix $\boldsymbol{A}$ of the implicit IB-LBM, while keeping its consistency. Kang and Hassan [75] provided an exhaustive overview of the multi direct-forcing IB-LBM. Since the underlying idea is similar to that of the implicit IB-LBM, we only provide the algorithm and a few comments.

Instead of constructing and inverting a large matrix $\boldsymbol{A}$, the multi direct-forcing method relies on an iterative approach to satisfy the no-slip condition at all markers $\boldsymbol{r}_j$ simultaneously. Again, the underlying idea is to take advantage of the velocity correction in (11.47). The algorithm of the multi direct-forcing approach can be summarised as follows:

1. Set iteration counter m to 0.
2. Stream the populations to obtain $f_i(\boldsymbol{x})$ and compute the density and uncorrected velocity from $\rho\boldsymbol{u}^{(m)} = \sum_i f_i \boldsymbol{c}_i$.
3. Interpolate $\boldsymbol{u}^{(m)}(\boldsymbol{x})$ at the boundary marker locations $\boldsymbol{r}_j$ to obtain $\boldsymbol{u}^{(m)}(\boldsymbol{r}_j)$.
4. Increment iteration counter m by 1.
5. Compute the Lagrangian correction force from [75]

$$\boldsymbol{f}_j^{(m)} = 2\rho \frac{\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j) - \boldsymbol{u}^{(m-1)}(\boldsymbol{r}_j)}{\Delta t}. \tag{11.53}$$

6. Spread $\boldsymbol{f}_j^{(m)}$ to the Eulerian lattice to obtain $\boldsymbol{F}^{(m)}(\boldsymbol{x})$.
7. Correct previous Eulerian velocity according to

$$\boldsymbol{u}^{(m)}(\boldsymbol{x}) = \boldsymbol{u}^{(m-1)}(\boldsymbol{x}) + \frac{\boldsymbol{F}^{(m)}(\boldsymbol{x})\Delta t}{2\rho(\boldsymbol{x})}. \tag{11.54}$$

8. Repeat steps 3–7 until m reaches a pre-defined limit m_max or until $\boldsymbol{u}^{(m)}(\boldsymbol{r}_j)$ converges to $\boldsymbol{u}_\mathrm{b}(\boldsymbol{r}_j)$.
9. Use the total correction force

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{m=1}^{m_\mathrm{max}} \boldsymbol{F}^{(m)}(\boldsymbol{x}) \tag{11.55}$$

   in the collision step.
10. Go back to step 1 for the next time step.

Kang and Hassan [75] compared results of benchmark tests for different iteration numbers up to $m_{\max} = 20$. They found that $m_{\max} = 5$ is a reasonable compromise of accuracy and efficiency. Since the iteration involves only those lattice nodes close to the boundary, the additional computational cost is relatively low.

11.4.4.4 Explicit, Non-iterative Direct-Forcing IB-LBM

As pointed out by Kang and Hassan [75], a non-iterative direct-forcing scheme can be obtained as special case of the multi direct-forcing method in the previous section. Setting $m_{\max} = 1$ leads to a simple explicit scheme that does not require expensive matrix inversions or iterations. This special case is commonly denoted "direct-forcing" IB-LBM, although the implicit and iterative methods are, strictly speaking, also direct-forcing methods.[7]

There exist different flavours of non-iterative direct-forcing IB-LBM (see for example [79]). However, it is obvious that this method will generally not give results of a comparable accuracy and consistency compared to implicit or iterative schemes.

11.4.5 Explicit IBM for Deformable Boundaries

The first works utilising the deformable IB-LBM for flowing deformable red blood cells were published in 2007 by several groups [104–106]. The overall algorithm follows the layout described in Sect. 11.4.2. The step from the IBM algorithm for rigid boundaries as presented in Sect. 11.4.3 to deformable boundaries is straightforward. Instead of finding suitable penalty forces to keep the boundary deformation as small as possible, one has to use forces which arise from elastic surface stresses due to the (desired) deformation of the boundary. This requires two additional ingredients: (i) a constitutive model for the boundary deformation and (ii) a surface mesh (i.e. markers *and* their connectivity) to evaluate the boundary deformation.

Sui et al. [107] were the first to present a 3D model for elastic particles (capsules, red blood cells) in an LB simulation with well-defined constitutive behaviour and a finite-element method to find the elastic membrane forces. Krüger et al. [85] later investigated the effect of the choice of interpolation stencil and distance between neighbouring Lagrangian nodes on the deformation of a capsule in shear flow. The IB-LBM for elastic problems has been applied to, for example, viscous flow over a flexible sheet [96] and dense suspensions of red blood cells [108] (see also Fig. 11.16).

[7] Remember that "direct forcing" means that there are no free parameters, such as the elasticity κ of the explicit method in Sect. 11.4.3.

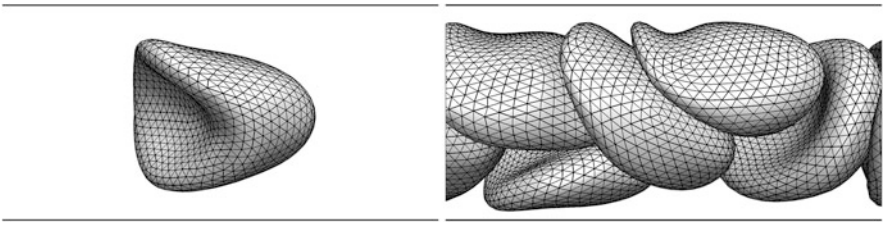

Fig. 11.16 Flow of a single (*left*) and multiple (*right*) red blood cells in a straight tube (indicated by *solid horizontal lines*) with circular cross-section. The tube diameter is slightly larger than that of an undeformed red blood cell. The Lagrangian mesh consists of 998 nodes and 2000 triangular elements. The simulations are based on the model presented in [108]

The IBM provides a major advantage over other boundary conditions for the LBM when it comes to deformable objects. Since IBM boundaries in the original implementation are intrinsically deformable, it is relatively simple to turn this presumed disadvantage into an advantage for problems where the deformability is actually desired. Allowing Lagrangian markers to move with the fluid and distributing forces to the fluid is the natural algorithm of the IBM and lends itself to problems where the fluid causes structure deformation and the structure "reacts" elastically. Applying any of the other boundary conditions presented in this chapter to deformable boundaries is significantly more difficult.

11.4.5.1 Constitutive Models

The *constitutive model* contains all the physics of the boundary deformation. Its choice is independent of the IBM algorithm itself and has to be defined by the user. In the end, the IBM expects the marker forces $\boldsymbol{f}_j$, but the IBM itself is unable to provide them. Boundaries are mostly considered hyperelastic (i.e. the dynamics can be fully described in terms of an energy density) or viscoelastic. In the former case, the marker forces depend only on the current deformation state, in the latter case the forces depend both on the deformation state and its rate of change.

There exists a large variety of hyperelastic and viscoelastic models for deformable boundaries. The problem of finding and implementing an appropriate constitutive model is highly problem-specific. We cannot delve into details here; this could easily fill a book on its own. The most commonly used hyperelastic models for red blood cells are briefly discussed in [107].

For simplicity, we will assume that hyperelastic models can be written in the form

$$\boldsymbol{f}_j(t) = \boldsymbol{f}_j(\{\boldsymbol{r}_k(t)\}) \tag{11.56}$$

and viscoelastic models as

$$\boldsymbol{f}_j(t) = \boldsymbol{f}_j(\{\boldsymbol{r}_k(t)\}, \{\dot{\boldsymbol{r}}_k(t)\}). \tag{11.57}$$

This means that the instantaneous marker forces are (arbitrarily complicated) functions of all current boundary marker positions and, if viscoelastic, of all current boundary marker velocities. Once these laws have been specified, they can be coded and used to find the forces $\boldsymbol{f}_j$ for a given deformation state at every time step.

Example 11.1 A simple hyperelastic constitutive model which can be used in 2D and 3D is (dropping the time dependence for simplicity)

$$\boldsymbol{f}_j(\{\boldsymbol{r}_k\}) = -\kappa \sum_{k \neq j} \frac{d_{jk} - d_{jk}^{(0)}}{d_{jk}^{(0)}} \frac{\boldsymbol{d}_{jk}}{d_{jk}}, \quad \boldsymbol{d}_{jk} = \boldsymbol{r}_k - \boldsymbol{r}_j, \quad d_{jk} = |\boldsymbol{d}_{jk}| \tag{11.58}$$

where κ is an elastic modulus, the sum runs over all next neighbours of marker j and $d_{jk}^{(0)}$ is the equilibrium distance between markers j and k. This example shows that not only the markers, but also their connectivity is an important part of the problem description. In many situations, additional constraints are necessary, for instance conservation of the total volume or surface of a boundary.

In most cases of hyperelastic boundaries one first defines an elastic energy density $\epsilon(\{\boldsymbol{r}_j(t)\})$. The force acting on node j can then be recovered by applying the principle of virtual work,

$$\boldsymbol{f}_j = \frac{\partial \epsilon(\{\boldsymbol{r}_k\})}{\partial \boldsymbol{r}_j} A_j, \tag{11.59}$$

where A_j is the area related to marker j, e.g. its Voronoi area. More details are provided in [85, 107].

11.4.6 Additional Variants and Similar Boundary Treatments

The previous sections cover the most prominent flavours of the IB-LBM. This is, however, not the end of the rope. There are more variations on the market, some of which we want to mention in the following.

There are a number of fluid-structure interaction approaches which share some features with the IBM (in particular the existence of off-lattice markers or a Lagrangian mesh and kernel functions for velocity interpolation and/or force spreading), but their algorithms reveal distinct differences. Schiller [109] recently revisited those algorithms and pointed out their mathematical similarity.

1. Ahlrichs and Dünweg [110] introduced a dissipative coupling method for LBM and molecular dynamics (MD), which has been further analysed by Caiazzo and Maddu [95] and recently reviewed by Dünweg and Ladd [111] and is used in the open-source package ESPResSo [112], mostly for polymer simulations. Lagrangian markers are allowed to move with a different velocity than the velocity of the fluid at the location of the marker (obtained by velocity interpolation). A finite slip velocity results in a drag force acting on the marker whose magnitude is controlled *via* a numerical drag coefficient. An equal but opposite force is exerted on the fluid by spreading it to the Eulerian lattice. Additionally, the markers may experience external or interaction forces. The marker update is treated by higher-order MD, which requires the introduction of another model parameter, a finite marker mass. An advantage of this approach is that the time step for the update of the markers is decoupled from the LB time step, which can be exploited to implement more stable time-integration schemes; a freedom which is not available for the conventional IB-LBM algorithm. Disadvantages are that the no-slip condition is not strictly satisfied and that two model parameters are required (drag coefficient and marker mass).
2. The momentum-exchange-based IB-LBM, as proposed for rigid boundaries [113, 114] and recently extended to flexible boundaries [115], uses a different approach to obtain the force density acting on the fluid. The basic idea is to interpolate the LB populations (rather than the velocity) to find their value at the location of the Lagrangian markers. The bounce-back scheme is then applied on the Lagrangian mesh to find the momentum exchange and therefore the marker force $\boldsymbol{f}_j$ which is then distributed to the lattice *via* the standard force spreading operation. As a consequence, there is no need for user-defined penalty parameters (for rigid boundaries) and the markers can move independently of the fluid motion. However, this may lead to a violation of the no-slip condition.[8]
3. Wu and Aidun [116] proposed the so-called *external-boundary force* (EBF) which can be used both as alternative to the direct-forcing IBM for rigid boundaries and for deformable objects. Similar to the other two examples above, the most notable difference to the IBM is that the markers are not directly advected by the fluid. Instead, a relative slip velocity is permitted which is counteracted by a fluid-solid interaction force, which is essentially a penalty force. Note that this force does not require a free parameter like the dissipative

[8] While the authors of [113] use bounce-back to obtain the momentum exchange at the boundary markers, the populations on the lattice are not directly affected by this bounce-back procedure. Although the momentum exchange is correctly obtained, there is no strong mechanism enforcing the local no-slip condition. Therefore, streamlines may penetrate the boundary.

coupling does. Also here, the allowance of a relative slip velocity may lead to problems with the no-slip condition.

Concluding, we can say that, although it is desirable to decouple the motion of the marker points from the fluid motion (as this allows higher-order and therefore potentially more stable time integration schemes), the no-slip condition at the boundary cannot be strictly enforced at the same time. The reason is that all the methods discussed above [110, 113, 115, 116] are explicit with respect to the fluid velocity computation. Still, if the precise realisation of the no-slip condition is not the primary goal, the above methods are attractive alternatives and overcome some of the disadvantages of the more conventional IB schemes.

It is also worth mentioning that the IBM can be used to model thermal boundary conditions. In 2010, Jeong et al. [117] combined the IBM with a thermal LBM to simulate flows around bluff bodies with heat transfer. Seta [118] later improved the thermal IB-LBM by analysing the governing equations through a Chapman-Enskog analysis. Another IBM variant that has apparently not yet been combined with the LBM is the so-called penalty IBM (p-IBM) [119–121] for flexible boundaries. It involves two set of Lagrangian markers: one interacting with the fluid, the other used for the calculation of the Lagrangian forces. Both marker sets are coupled by springs generating penalty forces.

11.5 Concluding Remarks

The number of available boundary conditions for the LBM is overwhelming, and it can be a daunting task to grasp the implications of those schemes. In the following we list a series of publications which provide comparative studies of boundary conditions for curved geometries. This should help to understand the relative performance of certain boundary methods for a given flow geometry.

- Ginzburg and d'Humières [17] compared simple (Sect. 11.2.1) and interpolated bounce-back (Sect. 11.2.2) with the multireflection method [17] in a number of stationary situations (inclined Couette and Poiseuille flows, flow over single cylinder and array of cylinders, impulsively started cylinder and moving sphere in a cylinder). They conclude that the multireflection method is more accurate than the linear interpolated bounce-back method.
- Pan et al. [12] compared the performance of simple bounce-back, interpolated bounce-back (both linear and quadratic interpolations) and multireflection boundary conditions for porous media simulations. They also included an analysis of the effect of collision operator (BGK *vs.* MRT) on the permeability of idealised porous media. Their main finding is that the permeability is generally viscosity-dependent through an unphysical dependence on the relaxation rate(s). Especially the combination of BGK and simple bounce-back leads to a strongly increasing permeability with viscosity, an effect caused by the increasing slip velocity at the boundary. They conclude that the combination of simple bounce-

back with MRT is consistently better than with BGK. The reason is that the no-slip condition can be much better controlled in the MRT framework when the viscosity is large. Using interpolated rather than simple bounce-back can also improve the accuracy of the simulations.

- Peng and Luo [94] investigated the relative performance of interpolated bounce-back and a direct-forcing immersed boundary method (IBM, Sect. 11.4.4). They considered steady and unsteady flows about a stationary rigid cylinder in 2D. Their major findings are that both methods require roughly the same computing time, the interpolated bounce-back is more accurate, but that the IBM is easier to implement.
- Chen et al. [47] have combined the ghost method (Sect. 11.3.4) with the partially saturated method (PSM, Sect. 11.2.3) in order to remove spurious pressure oscillations. The immediate consequence is that the algorithm becomes significantly more complicated than Noble's and Torczynski's [46] original one: interpolations become necessary to find the ghost node properties and a treatment of fresh nodes is required. This is a clear disadvantage compared to the original method [46] where the fresh node problem is naturally avoided. The conclusion is that the combined method yields better results than the original PSM or the interpolated bounce-back method, especially for moving obstacles at smaller resolutions. According to Chen et al., PSM is recommended when code simplicity and efficiency are desired, while the combined method should be favoured for high-accuracy applications.
- Chen et al. [14] recently conducted a thorough comparison of three bounce-back schemes (standard, interpolated and unified interpolation), two IBM variants (explicit and implicit direct forcing) and three additional methods. The authors were primarily interested in acoustic problems involving sound wave generation from moving bodies due to the fresh node treatment. The authors found that the IBM is more suitable for moving boundaries than the interpolated bounce-back when fresh nodes are involved.
- Nash et al. [122] compared the accuracy of simple and interpolated bounce-back, the Guo-Zheng-Shi extrapolation method (GZS, Sect. 11.3.3) and the Junk-Yang method in non-grid-aligned Poiseuille, Womersley and Dean flows at moderate Reynolds numbers (up to 300). The authors found that the Junk-Yang method shows poor stability in the selected parameter range. The linear interpolated bounce-back and the GZS methods have comparable accuracy (with a second-order convergence) although the latter becomes unstable for the highest Reynolds numbers tested. For the situation of interest (flow in inclined channels with moderate Reynolds number), the authors conclude that interpolated bounce-back is the best all-around option, although simple bounce-back (despite its first-order convergence) may be the method of choice when code development time is at a premium.

Everything said up to this point applies to *rigid* boundaries. It seems that the IBM and its related methods is still the most convenient approach for *deformable* boundaries (cf. Sects. 11.4.5 and 11.4.6). The reason is that all other boundary

conditions presented in this chapter require an accurate *local* momentum exchange algorithm to compute the local stresses in the deformable material.[9] This is normally a very challenging and expensive problem that is elegantly circumvented by the IBM.

We can generally conclude that all existing boundary conditions claim their own compromise of accuracy, stability and efficiency/ease of implementation. Furthermore, some boundary conditions perform better in stationary situations, others when the boundaries are moving. It is up to the user to identify the requirements before choosing one of the many available boundary conditions. There is no best boundary treatment for all possible scenarios. We hope that this chapter sheds some light on the plethora of boundary conditions and helps the reader to find a suitable scheme for a given problem.

References

1. S. Haeri, J.S. Shrimpton, Int. J. Multiphas. Flow **40**, 38 (2012)
2. X. He, G. Doolen, J. Comput. Phys. **134**, 306 (1997)
3. P. Lallemand, L.S. Luo, Phys. Rev. E **61**(6), 6546 (2000)
4. T. Lee, C.L. Lin, J. Comput. Phys. **171**(1), 336 (2001)
5. Z. Guo, T.S. Zhao, Phys. Rev. E **67**(6), 066709 (2003)
6. N. Rossi, S. Ubertini, G. Bella, S. Succi, Int. J. Numer. Meth. Fluids **49**(6), 619 (2005)
7. H. Yoshida, M. Nagaoka, J. Comput. Phys. **257, Part A**, 884 (2014)
8. R. Cornubert, D. d'Humières, D. Levermore, Physica D **47**, 241 (1991)
9. I. Ginzburg, P.M. Adler, J. Phys. II France **4**(2), 191 (1994)
10. A.J.C. Ladd, J. Fluid Mech. **271**, 285 (1994)
11. M. Bouzidi, M. Firdaouss, P. Lallemand, Phys. Fluids **13**, 3452 (2001)
12. C. Pan, L.S. Luo, C.T. Miller, Comput. Fluids **35**(8-9), 898 (2006)
13. O.E. Strack, B.K. Cook, Int. J. Numer. Meth. Fluids **55**(2), 103 (2007)
14. L. Chen, Y. Yu, J. Lu, G. Hou, Int. J. Numer. Meth. Fluids **74**(6), 439 (2014)
15. S. Khirevich, I. Ginzburg, U. Tallarek, J. Comp. Phys. **281**, 708 (2015)
16. M.O. Bernabeu, M.L. Jones, J.H. Nielsen, T. Krüger, R.W. Nash, D. Groen, S. Schmieschek, J. Hetherington, H. Gerhardt, C.A. Franco, P.V. Coveney, J. R. Soc. Interface **11**(99), 20140543 (2014)
17. I. Ginzburg, D. d'Humières, Phys. Rev. E **68**, 066614 (2003)
18. B. Chun, A.J.C. Ladd, Phys. Rev. E **75**, 066705 (2007)
19. X. He, Q. Zou, L.S. Luo, M. Dembo, J. Stat. Phys. **87**(1–2), 115 (1997)
20. I. Ginzburg, D. d'Humières, J. Stat. Phys. **84**, 927 (1996)
21. A.J.C. Ladd, Phys. Rev. Lett. **70**(9), 1339 (1993)
22. A.J.C. Ladd, J. Fluid Mech. **271**, 311 (1994)
23. A.J.C. Ladd, R. Verberg, J. Stat. Phys. **104**(5–6), 1191 (2001)
24. C.K. Aidun, J.R. Clausen, Annu. Rev. Fluid Mech. **42**, 439 (2010)
25. N.Q. Nguyen, A.J.C. Ladd, Phys. Rev. E **66**(4), 046708 (2002)
26. E.J. Ding, C.K. Aidun, J. Stat. Phys. **112**(3–4), 685 (2003)
27. C.K. Aidun, Y. Lu, E.J. Ding, J. Fluid Mech. **373**, 287 (1998)

[9] For rigid objects it is sufficient to compute the total momentum and angular momentum transfer. Soft objects, however, deform locally. This local deformation depends on the local stresses.

28. D. Qi, J. Fluid Mech. **385**, 41 (1999)
29. X. Yin, G. Le, J. Zhang, Phys. Rev. E **86**(2), 026701 (2012)
30. E. Lorenz, A. Caiazzo, A.G. Hoekstra, Phys. Rev. E **79**(3), 036705 (2009)
31. J.R. Clausen, C.K. Aidun, Int. J. Multiphas. Flow **35**(4), 307 (2009)
32. B. Wen, C. Zhang, Y. Tu, C. Wang, H. Fang, J. Comput. Phys. **266**, 161 (2014)
33. O. Behrend, Phys. Rev. E **52**(1), 1164 (1995)
34. M.A. Gallivan, D.R. Noble, J.G. Georgiadis, R.O. Buckius, Int. J. Numer. Meth. Fluids **25**(3), 249–263 (1997)
35. Y. Han, P.A. Cundall, Int. J. Numer. Meth. Fluids **67**(3), 314–327 (2011)
36. P.H. Kao, R.J. Yang, J. Comput. Phys. **227**(11), 5671 (2008)
37. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 427 (2008)
38. I. Ginzburg, F. Verhaeghe, D. d'Humières, Commun. Comput. Phys. **3**, 519 (2008)
39. X. Descovich, G. Pontrelli, S. Melchionna, S. Succi, S. Wassertheurer, Int. J. Mod. Phys. C **24**(05), 1350030 (2013)
40. P. Lallemand, L.S. Luo, J. Comput. Phys. **184**(2), 406 (2003)
41. D. Yu, R. Mei, W. Shyy, in *41st Aerospace Sciences Meeting and Exhibit*, 2003-953 (AIAA, New York, 2003)
42. X. Yin, J. Zhang, J. Comput. Phys. **231**(11), 4295 (2012)
43. O. Dardis, J. McCloskey, Phys. Rev. E **57**(4), 4834 (1998)
44. S.D.C. Walsh, H. Burwinkle, M.O. Saar, Comput. Geosci. **35**(6), 1186 (2009)
45. J. Zhu, J. Ma, Adv. Water Resour. **56**, 61 (2013)
46. D.R. Noble, J.R. Torczynski, Int. J. Mod. Phys. C **09**(08), 1189 (1998)
47. L. Chen, Y. Yu, G. Hou, Phys. Rev. E **87**(5), 053306 (2013)
48. I. Ginzburg, Adv. Water Resour. **88**, 241 (2016)
49. I. Ginzburg, G. Silva, L. Talon, Phys. Rev. E **91**, 023307 (2015)
50. H. Yoshida, H. Hayashi, J. Stat. Phys. **155**, 277 (2014)
51. G. Zhou, L. Wang, X. Wang, W. Ge, Phys. Rev. E **84**(6), 066701 (2011)
52. H. Yu, X. Chen, Z. Wang, D. Deep, E. Lima, Y. Zhao, S.D. Teague, Phys. Rev. E **89**(6), 063304 (2014)
53. R. Mei, L.S. Luo, P. Lallemand, D. d'Humières, Comput. Fluids **35**(8-9), 855 (2006)
54. G. Pontrelli, C.S. König, I. Halliday, T.J. Spencer, M.W. Collins, Q. Long, S. Succi, Med. Eng. Phys. **33**(7), 832 (2011)
55. B. Stahl, B. Chopard, J. Latt, Comput. Fluids **39**(9), 1625–1633 (2010)
56. M. Matyka, Z. Koza, Ł. Mirosław, Comput. Fluids **73**, 115 (2013)
57. X. Kang, Z. Dun, Int. J. Mod. Phys. C p. 1450057 (2014)
58. R. Mei, L.S. Luo, W. Shyy, J. Comput. Phys. **155**(2), 307 (1999)
59. B. Wen, H. Li, C. Zhang, H. Fang, Phys. Rev. E **85**(1), 016704 (2012)
60. Z.L. Guo, C.G. Zheng, B.C. Shi, Phys. Fluids **14**, 2007 (2002)
61. A. Tiwari, S.P. Vanka, Int. J. Numer. Meth. Fluids **69**(2), 481 (2012)
62. O.R. Mohammadipoor, H. Niazmand, S.A. Mirbozorgi, Phys. Rev. E **89**(1), 013309 (2014)
63. J.C.G. Verschaeve, B. Müller, J. Comput. Phys. **229**, 6781 (2010)
64. J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Phys. Rev. E **77**(5), 056703 (2008)
65. O. Filippova, D. Hänel, J. Comput. Phys. **147**, 219 (1998)
66. R. Mei, D. Yu, W. Shyy, L.S. Luo, Phys. Rev. E **65**(4), 041203 (2002)
67. J. Bao, P. Yuan, L. Schaefer, J. Comput. Phys. **227**(18), 8472 (2008)
68. R. Khazaeli, S. Mortazavi, M. Ashrafizaadeh, J. Comput. Phys. **250**, 126 (2013)
69. Q. Zou, X. He, Phys. Fluids **9**, 1591 (1997)
70. N. Pellerin, S. Leclaire, M. Reggio, Comput. Fluids **101**, 126 (2014)
71. C.S. Peskin, Flow patterns around heart valves: A digital computer method for solving the equations of motion. Ph.D. thesis, Sue Golding Graduate Division of Medical Sciences, Albert Einstein College of Medicine, Yeshiva University (1972)
72. C.S. Peskin, J. Comput. Phys. **25**(3), 220 (1977)
73. C.S. Peskin, Acta Numerica **11**, 479–517 (2002)
74. Z.G. Feng, E.E. Michaelides, J. Comput. Phys. **195**(2), 602 (2004)

75. S.K. Kang, Y.A. Hassan, Int. J. Numer. Meth. Fluids **66**(9), 1132 (2011)
76. Y. Cheng, L. Zhu, C. Zhang, Commun. Comput. Phys. **16**(1), 136 (2014)
77. R. Mittal, G. Iaccarino, Annu. Rev. Fluid Mech. **37**, 239 (2005)
78. J. Lu, H. Han, B. Shi, Z. Guo, Phys. Rev. E **85**(1), 016711 (2012)
79. T. Seta, R. Rojas, K. Hayashi, A. Tomiyama, Phys. Rev. E **89**(2), 023307 (2014)
80. B.E. Griffith, X. Luo, D.M. McQueen, C.S. Peskin, Int. J. Appl. Mech. **01**, 137 (2009)
81. X. Wang, L.T. Zhang, Comput. Mech. **45**(4), 321 (2010)
82. X. Yang, X. Zhang, Z. Li, G.W. He, J. Comput. Phys. **228**(20), 7821 (2009)
83. Z. Guo, C. Zheng, B. Shi, Phys. Rev. E **65**, 46308 (2002)
84. S.K. Doddi, P. Bagchi, Int. J. Multiphas. Flow **34**(10), 966 (2008)
85. T. Krüger, F. Varnik, D. Raabe, Comput. Method. Appl. **61**(12), 3485 (2011)
86. G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.J. Jan, J. Comput. Phys. **169**(2), 708 (2001)
87. K. Suzuki, T. Inamuro, Comput. Fluids **49**(1), 173 (2011)
88. D. Nie, J. Lin, Commun. Comput. Phys. **7**(3), 544 (2010)
89. S.K. Doddi, P. Bagchi, Phys. Rev. E **79**(4), 046318 (2009)
90. S. Ramanujan, C. Pozrikidis, J. Fluid Mech. **361**, 117 (1998)
91. Z.G. Feng, E.E. Michaelides, Comput. Fluids **38**(2), 370 (2009)
92. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org
93. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://www.geuz.org/gmsh
94. Y. Peng, L.S. Luo, Prog. Comput. Fluid Dyn. **8**(1), 156 (2008)
95. A. Caiazzo, S. Maddu, Comput. Math. Appl. **58**(5), 930 (2009)
96. L. Zhu, G. He, S. Wang, L. Miller, X. Zhang, Q. You, S. Fang, Comput. Math. Appl. **61**(12), 3506 (2011)
97. Q. Zhou, L.S. Fan, J. Comput. Phys. **268**, 269 (2014)
98. O. Shardt, J.J. Derksen, Int. J. Multiphase Flow **47**, 25 (2012)
99. G. Le, J. Zhang, Phys. Rev. E **79**(2), 026701 (2009)
100. Z.G. Feng, E.E. Michaelides, J. Comput. Phys. **202**(1), 20 (2005)
101. J. Wu, C. Shu, J. Comput. Phys. **228**(6), 1963 (2009)
102. J. Wu, C. Shu, Y.H. Zhang, Int. J. Numer. Meth. Fluids **62**(3), 327 (2010)
103. J. Wu, C. Shu, Int. J. Numer. Meth. Fluids **68**(8), 977 (2012)
104. P. Bagchi, Biophys. J. **92**(6), 1858 (2007)
105. M.M. Dupin, I. Halliday, C.M. Care, L. Alboul, L.L. Munn, Phys. Rev. E **75**(6), 066707 (2007)
106. J. Zhang, P.C. Johnson, A.S. Popel, Phys. Biol. **4**(4), 285 (2007)
107. Y. Sui, Y. Chew, P. Roy, H. Low, J. Comput. Phys. **227**(12), 6351 (2008)
108. T. Krüger, M. Gross, D. Raabe, F. Varnik, Soft Matter **9**(37), 9008 (2013)
109. U.D. Schiller, Comput. Phys. Commun. **185**(10), 2586 (2014)
110. P. Ahlrichs, B. Dünweg, Int. J. Mod. Phys. C **09**(08), 1429 (1998)
111. B. Dünweg, A.J.C. Ladd, in *Advances in Polymer Science* (Springer, Berlin, Heidelberg, 2008), pp. 1–78
112. I. Cimrák, M. Gusenbauer, I. Jančigová, Comput. Phys. Commun. **185**(3), 900 (2014)
113. X.D. Niu, C. Shu, Y.T. Chew, Y. Peng, Phys. Lett. A **354**(3), 173 (2006)
114. Y. Hu, H. Yuan, S. Shu, X. Niu, M. Li, Comput. Math. Appl. **68**(3), 140 (2014)
115. H.Z. Yuan, X.D. Niu, S. Shu, M. Li, H. Yamaguchi, Comput. Math. Appl. **67**(5), 1039 (2014)
116. J. Wu, C.K. Aidun, Int. J. Numer. Meth. Fl. **62**(7), 765–783 (2009)
117. H.K. Jeong, H.S. Yoon, M.Y. Ha, M. Tsutahara, J. Comput. Phys. **229**(7), 2526 (2010)
118. T. Seta, Phys. Rev. E **87**(6), 063304 (2013)
119. Y. Kim, M.C. Lai, J. Comput. Phys. **229**(12), 4840 (2010)
120. W.X. Huang, C.B. Chang, H.J. Sung, J. Comput. Phys. **230**(12), 5061 (2011)
121. W.X. Huang, C.B. Chang, H.J. Sung, J. Comput. Phys. **231**(8), 3340 (2012)
122. R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, Phys. Rev. E **89**(2), 023303 (2014)