IMPLEMENTING THE LATTICE-BOLTZMANN METHOD

A RESEARCH ON BOUNDARY CONDITION TECHNIQUES

by

S.C. Wetstein

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Applied Physics

at the Delft University of Technology,
to be defended in 2014

Supervisor:	Dr. ir. M. Rohde	
Thesis committee:	Dr. ir. M. Rohde,	TU Delft
	.....,	TU Delft

TU Delft Delft University of Technology

ABSTRACT

The pebble-bed reactor is a type of high temperature reactor. In this reactor, fuel is contained in pebbles which are pyrolytic graphite spheres. These spheres are stacked in the reactor core and are cooled by helium flowing around them. Because the helium flows through a random stacking of pebbles there are void fractions and random coolant flows, creating coolant velocity and even more local temperature variations. It is important to have knowledge about the flow of helium to make sure the temperature cannot reach a to high level at certain points in the reactor core.

In this thesis an algorithm is written to simulate the flow around spheres with the lattice-Boltzmann method. With this method a fluid is modeled by particles, which can move in constraint directions on a grid. With each time step the particles move to a neighboring node after which collisions take place, in which the particles redistribute their velocities. On the macroscopic scale, the flow of a fluid can be simulated with this method.

The focus of this thesis has been on a range of available boundary condition techniques in the lattice-Boltzmann method. No-slip boundary conditions are implemented meaning that the velocity at the encountered object wall is zero. The boundary conditions implemented are the simple half-way bounce-back boundary technique, linear and quadratic interpolation techniques and the multireflection boundary condition technique.

It is found that the half-way bounce-back boundary condition is almost as accurate as the interpolation techniques on a relatively fine grid. Since the half-way bounce-back technique is much easier to implement than the interpolation techniques using this technique is recommended. However, a finer grid does lead to an exponentially longer computation time. That is why, when time is of the essence, there is the option to do simulations on a coarser grid. On a coarser grid the linear and quadratic interpolation techniques are more accurate than the half-way bounce-back technique when simulating fluid flow.

The multireflection boundary condition was not implemented correctly in this thesis. Recommendations for reaching correct implementation are given.

CONTENTS

Nomenclature **3**

1 Introduction **7**
1.1 World energy problem . . . 7
1.2 The pebble-bed reactor . . . 7
1.3 The lattice-Boltzmann method . . . 8
1.4 Outline of the thesis . . . 9

2 Theory **11**
2.1 The lattice-Boltzmann method . . . 11
2.1.1 The lattice-Boltzmann equation . . . 11
2.1.2 The single-relaxation lattice-Boltzmann equation . . . 11
2.1.3 The equilibrium distribution . . . 12
2.1.4 Body force . . . 13
2.2 Boundary conditions . . . 13
2.2.1 Periodic boundary conditions . . . 13
2.2.2 Bounce-back boundary conditions . . . 13
2.2.3 Interpolation boundary conditions . . . 14

3 LBM models applied **17**
3.1 The D2Q9 model with half-way bounce-back . . . 17
3.2 The D3Q19 model . . . 18

4 Physical consistency **21**

5 Results and discussion **23**
5.1 Flow patterns . . . 23
5.2 Boundary techniques compared at different sphere refinements . . . 23
5.3 Boundary techniques compared at different ϕ . . . 26
5.4 Sphere refinement effects for the different boundary conditions . . . 27
5.5 Boundary techniques compared in computational efficiency . . . 28

6 Conclusions and recommendations **33**

Bibliography **35**

A Appendix **37**
A.1 D2Q9 Half-way bounce-back code . . . 37
A.2 D3Q19 Half-way bounce-back code . . . 39
A.3 MATLAB formula for finding all relevant line-sphere intersections . . . 41
A.4 D3Q19 Linear interpolation code . . . 43
A.5 D3Q19 Quadratic interpolation code . . . 45
A.6 D3Q19 Multireflection code . . . 48

NOMENCLATURE

Symbol	Description	Unit
A	Frontal area of the sphere	ls^2
C_d	Drag coefficient	-
c_s	Speed of sound	$ls \cdot lt^{-1}$
d	Distance from fluid node to boundary	ls
D	Diameter of the sphere	ls
$\vec{e}_q$	Lattice velocity in direction q	$ls \cdot lt^{-1}$
ε	Relative error in the Reynolds number	-
$\vec{F}$	Body force	$ls \cdot lt^{-2}$
$\vec{F}_d$	Drag force	$ls \cdot lt^{-2}$
$\vec{F}_{d,Has}$	Analytical drag force	$ls \cdot lt^{-2}$
$\vec{g}$	Magnitude of acceleration	$ls \cdot lt^{-2}$
n_q	Local mass distribution	-
n_q^*	Post collision mass distribution	-
$n_{q'}$	Local mass distribution in the direction opposite to q	-
n_q^{eq}	Equilibrium mass distribution	-
p	Pressure	$ls^{-1 \cdot lt^{-2}}$
ρ	Density	ls^{-3}
$\vec{r}_j$	Spatial position	ls
Re	Reynolds number	-
Δt	Discrete time step	lt
τ	Relaxation time constant of the BGK method	-
$\vec{u}$	Fluid velocity	$ls \cdot lt^{-1}$
$\vec{u}_b$	Velocity of the boundary	$ls \cdot lt^{-1}$
V_f	Volume of fluid	ls^3
ν	Kinematic viscosity	$ls^2 \cdot lt^{-1}$
ω	Weighing factor	-
Δx	Grid spacing	ls
Ω_q	Collision operator in direction q	-
ϕ	Volume fraction of spheres	-

1

INTRODUCTION

In the past twenty years, techniques for modeling boundary conditions in the Lattice-Boltzmann Method (LBM) have been a hot topic. The easiness of implementation and the accuracy were most analyzed. One of the earliest techniques was the bounce-back method which is very easy to implement, but only shows second-order accuracy with respect to spatial resolution under certain conditions [1],[2] and [3]. Later treatments suggested interpolation boundaries using a various amount of neighboring nodes [4],[5],[6] and [7]. These boundaries were designed to have a second-order accuracy. Lots of comparative studies were done on the interpolation boundary techniques. The multireflection boundary condition [8] in particular, turned out to deliver a lower magnitude of error than others by making a greater use of local velocities at neighboring nodes [9]. However, there are also some downsides to this technique, in particular on its convergence behavior. In this thesis we compare bounce-back, linear interpolation, quadratic interpolation and multireflection boundary techniques in the LBM. First, we will explain why we need these techniques and we will introduce an application where these techniques could be used, the pebble-bed reactor.

1.1. WORLD ENERGY PROBLEM

In the past decades the energy demand has strongly increased and, according to the International Energy Agency, this demand is expected to keep growing. They predict that in 2035 the world energy demand will have grown by one-third. This increase will be mostly (for 90%) caused by emerging economies in India and South-East Asia [10].

In 2012 86% of all energy was generated by burning fossil fuels (i.e. coal, oil and gas), as can be seen in figure 1.1. Fossil fuels are used so much because they are easier to use than any other energy source. A problem with using fossil fuels is that there is only a limited amount available and it is running out. Another problem with fossil fuels is that in the burning process pollutants are released, like greenhouse and toxic gases. It is clear that the world needs sustainable energy sources that can meet the growing world energy demand.

Nuclear energy can be part of the alternative energy supply. As we can see in figure 1.1 in 2012 only 5% of the world primary energy supply came from nuclear energy. This is because some aspects of nuclear energy, like nuclear waste and political opposition due to possible accidents, make the energy source look unattractive to the public. In order to identify nuclear energy as a true alternative energy, a sustainable nuclear energy technology platform was established in the European Union. This platform created new sustainability criteria for enhanced safety in nuclear reactors. To be able to meet these criteria new types of reactors were designed.

1.2. THE PEBBLE-BED REACTOR

The High Temperature Reactor (HTR) is one of those new nuclear reactor designs, mostly interesting because of the inherent safety of the reactor. The HTR has two main configuration types: the prismatic block core and the pebble-bed configuration, in this thesis we focus on the pebble-bed reactor. In this kind of reactor fuel is contained in so called pebbles. These pebbles are pyrolytic graphite spheres with a diameter of six centimeters. The core of the reactor is made of hundreds of thousands of these pebbles forming a randomly

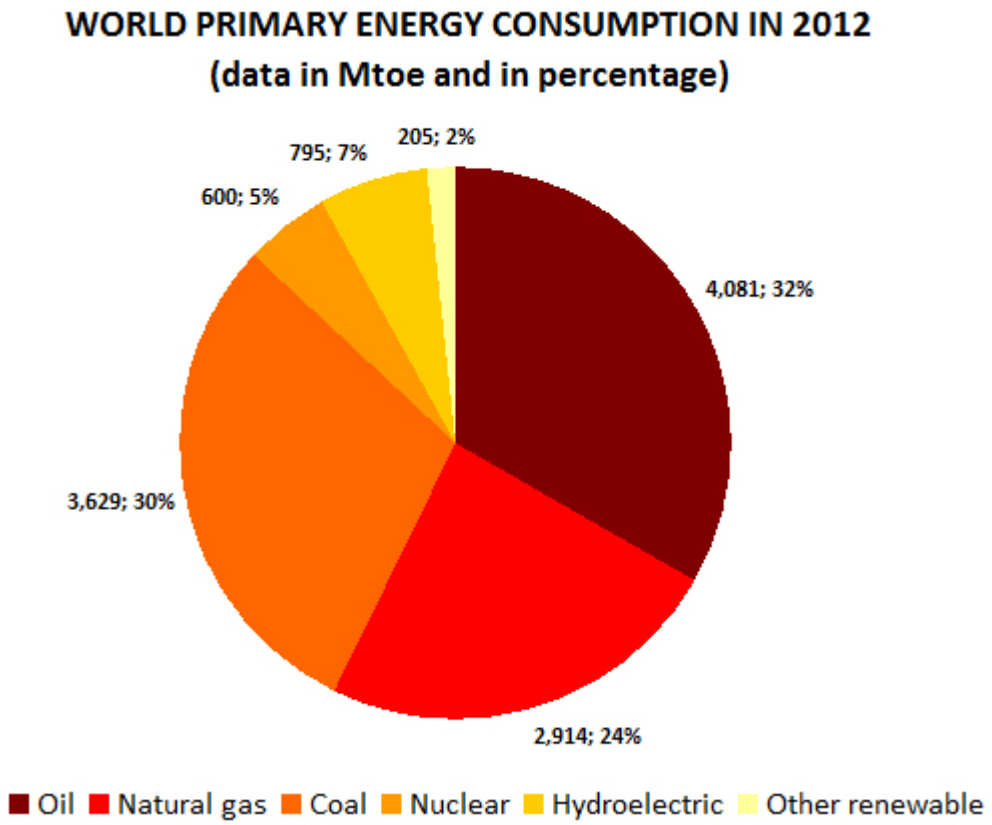


Figure 1.1: The world total primary energy supply in 2012. Source:[10]

packed bed, called the pebble-bed. Around the core there is a cylindrical graphite wall, the core is depicted in figure 1.2.

The pebble-bed reactor can be refueled while in operation by adding pebbles on the top and removing used pebbles at the bottom. A mound of pebbles forms below the drop point and the pebbles roll of the mound until they reach a stable position. This stable position is therefore somewhat randomly determined, and that causes the problem that hot spots may develop. Hot spots are clusters of highly reactive pebbles, which can form in regions of high thermal neutron flux, so that the local temperature can get very high. The core of the reactor is cooled by flowing helium through the randomly packed pebble-bed. The outlet temperature of the helium gas can be up to 1000°C. Because the helium flows through a random stacking of pebbles there are void fractions and random coolant flows, creating coolant velocity and even more local temperature variations. The temperature at each point in the core of the reactor is therefore dependent on the stacking of the pebbles, and can only be statistically calculated. This is were the LBM comes in. The flow of helium gas around the pebbles can be computationally modeled with the LBM. However, for this to be effective, we need the right boundary conditions to describe the complex geometry that a sphere is on a square grid.

1.3. THE LATTICE-BOLTZMANN METHOD

The lattice-Boltzmann method is a relatively new computational fluid dynamics (CFD) method for simulating fluid flow. It was introduced in 1988 by McNamara and Zanetti [12] to overcome the drawbacks of the lattice gas cellular automata. In a lattice gas automaton (LGA), a fluid can be considered as a collection of discrete particles which interact with each other only via certain rules. The LGA was introduced by Hardy et al. [13] in 1976, here discrete particles were residing on a square lattice. Later the use of a hexagonal lattice was introduced, making it possible to establish isotropy. Simulations of the LGA method unfortunately showed a lot of noise, due to the use of discrete particles. McNamara and Zanetti [12] proposed to average the set of basic LGA rules over a fictitious number of lattice nodes, formulating the LGA in terms of particle densities. This formulation is the lattice-Boltzmann equation, and produces much less noisy results.

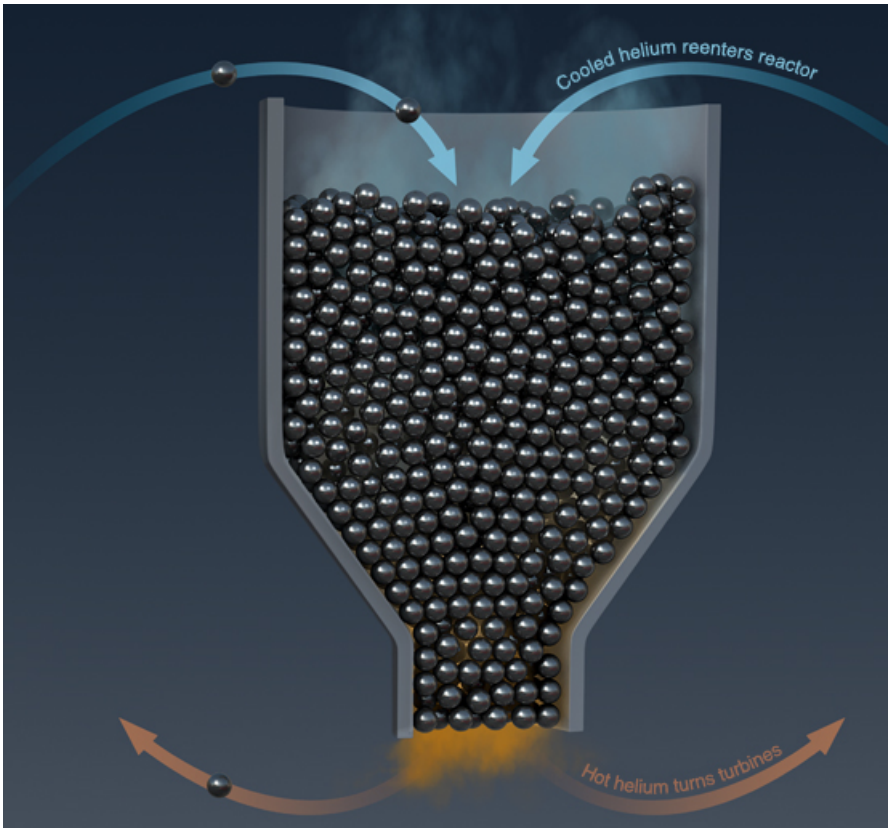


Figure 1.2: A schematic representation of the core of a pebble-bed reactor, directly cooled by helium. Source:[11]

In other CFD methods the Navier-Stokes equations solve mass, momentum and energy conservation equations on discrete nodes, elements or volumes. So the nonlinear partial differential equations are converted into a set of nonlinear algebraic equations, which are solved iteratively [14]. In the LBM a fluid is modeled by particles, which can move in a constraint number of directions, on a grid. This is a conceptually very different approach. With each time step the particles move to a neighboring node after which collisions take place, in which the particles redistribute their velocities in a certain way. The LBM is simpler than most other methods, since it is an explicit first-order discretized scheme (though second-order accurate in space and time). This technique is relatively easy to parallelize and can describe complex boundaries with short algorithms, which dramatically shortens computation time.

1.4. OUTLINE OF THE THESIS

In this thesis we will examine different boundary conditions in the Lattice-Boltzmann Method. We begin with basic theory of the LBM and boundary condition techniques. After the theory we will proceed by explaining the different LBM models used here and how we applied them. Next we will explain how we checked the physical consistency of the results and continue with the results on the different boundary conditions and a discussion of them. Finally, we will draw our conclusions on the boundary techniques implemented.

2

THEORY

2.1. THE LATTICE-BOLTZMANN METHOD

The Lattice-Boltzmann Method is a numerical method for fluid simulation. Macroscopic fluid flow properties are described by analyzing microscopic processes. These microscopic and macroscopic properties obey the Navier-Stokes equation. The lattice-Boltzmann method can, using the Chapman-Enskog procedure, give us the Navier-Stokes equation with higher order terms

$$\delta_t \vec{u} + (\vec{u}\cdot\nabla)\,\vec{u} = \frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u} + \frac{1}{3}\Delta t\nu^2\left(\tau - \frac{1}{2}\right)\nabla\left(\nabla\cdot\rho\vec{u}\right) + \mathcal{O}\left(u^3\right), \tag{2.1}$$

as shown by Bao and Meskas [15].

The LBM is used because the method has a simple algorithm which needs little computation time and is still second-order accurate. The method analyzes flow by solving the Lattice-Boltzmann equation. For this method a lattice unit conversion should be taken into account. In our work ls is the basic unit for lattice spacing, if the domain has a length L and N grid nodes, than the lattice spacing is defined as $ls = \frac{L}{N}$. The discrete time unit is given as $lt = \frac{ls}{c_s}$, where c_s is the speed of sound. All variables used in this thesis can be found in the nomenclature where the corresponding unit is given.

2.1.1. THE LATTICE-BOLTZMANN EQUATION

The Lattice-Boltzmann equation is a finite difference equation that describes the particle distribution function, which represents flow velocities of virtual particles. The function is defined as $n_q(\vec{r}_j, t)$, which denotes the mass density at location $(\vec{r}_j, t)$ moving in direction q. These virtual particles move to location $(\vec{r}_j + \vec{e}_q\Delta t)$, where $\vec{e}_q$ is the discretized particle velocity, after time Δt has elapsed. This moving of virtual particles represents the convection of the fluid. In the absence of collisions the equation is defined as

$$n_q(\vec{r}_j + \vec{e}_q\Delta t, t + \Delta t) = n_q(\vec{r}_j, t). \tag{2.2}$$

The Lattice-Boltzmann equation in which collisions between the particles are included uses a collision function $\Omega_q(n)$. As we will see in the next section, the collision operator depends on a relaxation parameter, which determines the kinematic viscosity of the simulated fluid. Only collisions between priorily uncorrelated particles are taken into account.

$$n_q(\vec{r}_j + \vec{e}_q\Delta t, t + \Delta t) = n_q(\vec{r}_j, t) + \Omega_q(n) \tag{2.3}$$

Ω_q must satisfy conservation of mass and momentum at all lattice points:

$$\sum_{q=1}^{M}\Omega_q = 0, \qquad \sum_{q=1}^{M}\Omega_q\vec{e}_q = 0. \tag{2.4}$$

2.1.2. THE SINGLE-RELAXATION LATTICE-BOLTZMANN EQUATION

The density ρ and momentum density $\rho\vec{u}$ are defined as particle velocity moments of the distribution function, n_q,

$$\rho = \sum_{q=1}^{M} n_q, \qquad \rho\vec{u} = \sum_{q=1}^{M} n_q\vec{e}_q. \tag{2.5}$$

If only the physics in the long wavelength and low-frequency limit are of interest, the lattice spacing and the time increment Δt in equation (2.3) can be regarded as small parameters of the same order ϵ, according to Chen and Doolen [16]. Performing an expansion of n_q around the local equilibrium distribution function n_q^{eq} in the small parameter $\epsilon \ll 1$,

$$n_q = n_q^{eq} + \epsilon n_q^{(neq)}, \tag{2.6}$$

where n_q^{eq} depends on the local macroscopic density and momentum density and should satisfy:

$$\sum_{q=1}^{M} n_q^{eq} = \rho \qquad \sum_{q=1}^{M} n_q^{eq} \vec{e}_q = \rho \vec{u}. \tag{2.7}$$

In equation (2.6) $n_q^{(neq)}$ is the nonequilibrium distribution function and is defined as [17]

$$n_q^{(neq)} = n_q^{(1)} + \epsilon n_q^{(2)} + O(\epsilon^2), \tag{2.8}$$

where the superscripts (1) and (2) denote the order of the Mach number expansion. Similarly, if we use a Taylor expansion on the collision operator $\Omega_q(n)$ in n we get

$$\Omega_q(n) = \Omega_q(n^{eq}) + \frac{\partial \Omega_q(n^{eq})}{\partial n_p}\left(\epsilon n_p^{(1)}\right) + \frac{\partial \Omega_q(n^{eq})}{\partial n_p}\left(\epsilon^2 n_p^{(2)}\right) + O(\epsilon^2 n^2). \tag{2.9}$$

When $\epsilon \to 0$, we have $\Omega_q(n^{eq}) = 0$, more details in [17]. The linearized collision operator can therefore be written as

$$\Omega_q(n) = \frac{\partial \Omega_q(n^{eq})}{\partial n_p}\left(\epsilon n_p^{(1)} + \epsilon^2 n_p^{(2)}\right) = M_{qp}\left(n_p - n_p^{eq}\right), \tag{2.10}$$

where M_{qp} is the collision matrix which determines the scattering between particles moving from direction q to direction p [18]. Assuming that the matrix M_{qp} has one dominant eigenvalue, τ, we get the lattice BGK collision term [19] which is written as

$$\Omega_q(n) = -\frac{1}{\tau}\left(n_q - n_q^{eq}\right). \tag{2.11}$$

Now the lattice-Boltzmann equation (2.3) can be simplified to

$$n_q(\vec{r}_j + \vec{e}_q \Delta t, t + \Delta t) = n_q(\vec{r}_j, t) - \frac{n_q - n_q^{eq}}{\tau}. \tag{2.12}$$

This equation is called the single-relaxation approximation, or the LBGK equation. Here τ is the time scale in which the distribution function relaxes to its equilibrium state.

2.1.3. THE EQUILIBRIUM DISTRIBUTION

The local D-dimensional, where D is the spatial dimension, equilibrium distribution function can be derived from the D-dimensional Maxwell-Boltzmann equilibrium distribution [17]

$$n^{eq}(\vec{\vartheta}, \vec{\varphi}) = \rho\left(\frac{3}{2\pi v^2}\right)^{D/2} e^{-\frac{3}{2}\left(\vec{\vartheta} - \vec{\varphi}\right)^2}, \tag{2.13}$$

where $\vec{\vartheta} = \frac{\vec{e}}{v}$ is the normalized particle velocity and $\vec{\varphi} = \frac{\vec{u}}{v}$ is the normalized fluid velocity. Here $v = \sqrt{\frac{3kT}{m}}$ is the average root-mean-square particle velocity. All our simulations are in three dimensions, so D = 3. When considering each direction q separately and taking the Taylor expansion of the equilibrium distribution up to the second order in the fluid velocity, we get [17]

$$n_q^{eq} \approx \omega_q \rho\left(1 + 3\frac{\vec{e}_q \cdot \vec{u}}{v^2} + \frac{9}{2}\frac{\left(\vec{e}_q \cdot \vec{u}\right)^2}{v^4} - \frac{3}{2}\frac{\vec{u} \cdot \vec{u}}{v^2}\right). \tag{2.14}$$

Here $v = \frac{\Delta x}{\Delta t}$ and the factors ω_q depend on the underlying grid and thereby also on the spatial dimension. The expressions for the pressure and viscosity can be derived using the equilibrium distribution. The equations are

$$p = \rho c_s^2, \qquad \nu = \Delta t \cdot c_s^2\left(\tau - \frac{1}{2}\right), \tag{2.15}$$

where c_s is the speed of sound, the pressure does not depend on the velocity anymore and the fluid kinematic viscosity, ν, depends on the time scale in which the distribution function relaxes to its equilibrium state, τ.

2.1.4. BODY FORCE

The fluid flow used here experiences a body force. A body force acts throughout the volume of a body, examples of body forces are gravitation and electromagnetic forces. Here the force acts upon all grid nodes, so momentum is added during each time step. Therefore we can add a force term to equation 2.12, making it

$$n_q(\vec{r}_j + \vec{e}_q \Delta t, t + \Delta t) = n_q(\vec{r}_j, t) - \frac{n_q - n_q^{eq}}{\tau} + \omega_q \left(\vec{e}_q \cdot \vec{F}\right) \frac{\Delta t}{c_s^2}, \tag{2.16}$$

where ω_q represents a direction dependent weight factor originating from the lattice-Boltzmann schema. It depends on the spatial dimension and the underlying grid.

2.2. BOUNDARY CONDITIONS

Techniques within the lattice-Boltzmann framework that mimic boundary conditions are necessary because we have to do with flow (as a result of a differential equation). Since the nineties, there has been a lot of research into the behavior of various boundary conditions for the LBM. Accurate boundary conditions are very important to obtain accurate results for systems with a complex geometry, but also for systems with a simple geometry like a sphere. The boundary conditions needed for fluid flowing around a sphere are no-slip boundary conditions. We need the no-slip boundary condition because here the velocity at the wall is zero, treating the sphere as a solid object from which the particles bounce back. In this thesis, details for periodic, bounce-back, linear and quadratic interpolation following Bouzidi et al. [4] and multireflection boundary conditions following Ginzburg and d'Humières [8] will be given.

2.2.1. PERIODIC BOUNDARY CONDITIONS

With periodic boundary conditions the system becomes closed in the sense that the edges are being treated as if they are attached to the opposite side. An easy way of implementing periodic boundaries is by copying the boundary nodes from one side of the domain to ghost nodes on the other side of the domain and then letting the fluid propagate. Full periodic boundaries are also useful when simulating an infinite flow domain.

2.2.2. BOUNCE-BACK BOUNDARY CONDITIONS

Bounce-back boundaries consist of a distribution function bounce-back scheme used at walls and objects to obtain a no-slip velocity condition. These boundaries however do not ensure a perfect no-slip boundary condition, it has been noticed that the bounce-back scheme actually only gives first-order spatial accuracy in boundaries [6], [20]. These boundaries are created by designating a particular node as a solid obstacle. The solids are separated into two types, nodes that lie at the solid-fluid interface and solid nodes that do not contact fluid nodes. The last nodes no longer communicate with the boundary nodes. With the full-way bounce-back, upon collision with an obstacle, the fluid particles simply reverse their direction of motion as can be seen in figure 2.1 and which can be expressed as

$$n^*_{q'}(\vec{r}_j, t) = n_q(\vec{r}_j, t), \tag{2.17}$$

where q' denotes the direction opposite to q and n^*_q denotes the post collision population.

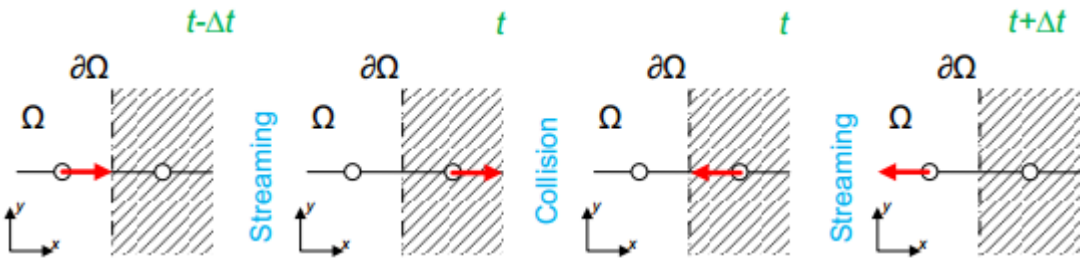


Figure 2.1: A schematic overview of full-way bounce-back. Here Ω represents the fluid and $\partial\Omega$ represents the boundary. In the propagation (or streaming) step a fluid population leaves the fluid node and reaches the boundary solid node. Then in the collision step its velocity is reversed and in the next streaming step the fluid population leaves the boundary solid node and gets back to the fluid node, only having a reversed direction of motion. This process takes two time steps. Source:[21].

In the half-way bounce-back rule, the boundary surface is assumed to lie halfway between a fluid node and a solid node. A fluid population leaving the fluid node $\vec{r}_j$ and encountering the solid node is reflected

and returns in one time step to its original location, where it now points in the opposite direction. This can be formulated as

$$n_{q'}(\vec{r}_j, t+\Delta t) = n^*_q(\vec{r}_j, t). \tag{2.18}$$

A graphic example of half-way bounce-back is portrayed in figure 2.2.

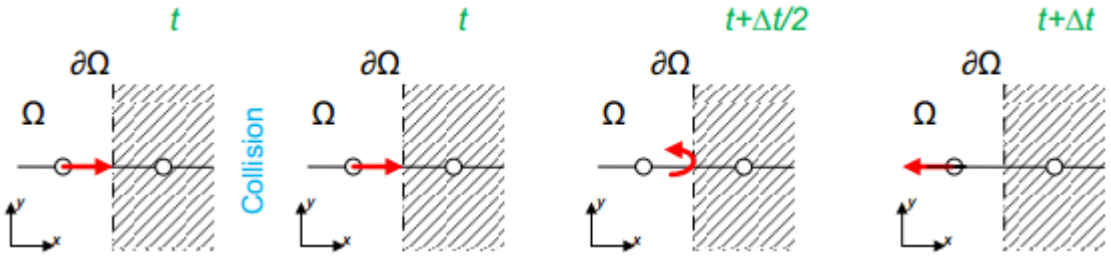


Figure 2.2: A schematic overview of half-way bounce-back. Here Ω represents the fluid and $\partial\Omega$ represents the boundary. In the collision step a fluid population leaving the fluid node reaches the boundary, which lies halfway between the fluid node and the solid node, at time $t + \frac{\Delta t}{2}$. Here the direction of propagation reverses. Next in the streaming step the fluid population moves back to the fluid node, only having a reversed direction of motion. This process takes only one time step. Source:[21].

As said before, the half-way bounce-back scheme is known to model a boundary which lies midway between boundary nodes and neighboring fluid nodes. Based on this interpretation, the approximate shapes of the spheres, circles since its portrayed in two dimensions, are shown in figure 2.3. It is apparent that the bounce-back boundary condition cannot directly model a general curvilinear surface but instead it uses a staircase shaped approximation of the surface.

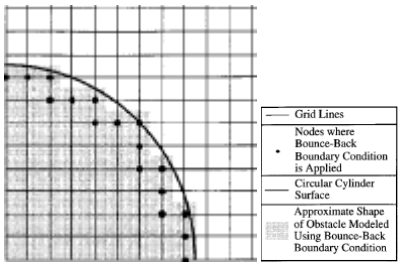


Figure 2.3: The approximate shape of a two dimensional sphere (a circle) when modeling it with the half-way bounce-back boundary condition. It is apparent that this boundary condition uses a staircase shaped approximation of the surface. Source:[20].

2.2.3. INTERPOLATION BOUNDARY CONDITIONS

As can be seen in figure 2.3, the solid surface is not always located midway between a fluid node and a boundary solid node, resulting in a first-order accurate method. Therefore a second-order boundary condition requires interpolation. In this thesis the focus lies on linear and quadratic interpolation schemes as proposed by Bouzidi et al. [4] and the multireflection scheme as proposed by Ginzburg and d'Humières [8]. The linear and quadratic interpolation schemes use different formulas depending on the location of the boundary. This is done in order to make sure that the distribution is always interpolated and never extrapolated, which ensures stability [22]. What is meant with the location of the boundary is made clear in figure 2.4.

If the wall is closer to the fluid node than to the solid node, so $d < \frac{1}{2}\Delta x$, an interpolated population is constructed farther away from the wall (at D) and then bounced back to end up on the fluid node near the wall (at A). In the other case of the wall being located closer to the solid node, so $d \geq \frac{1}{2}\Delta x$ the population of the fluid node (at A) is bounced back from the wall to end up at an intermediate location (at D). This intermediate population is combined with the neighboring population to interpolate to the fluid boundary node.
When making real computations, we first determine all cases in which the solid boundary is crossed. This is when we have a fluid node $\vec{r}_j$ such that $\vec{r}_j + \vec{e}_q\Delta t$ is a solid node. Since the geometries investigated in this paper are spheres we need to find all distances d from fluid nodes to the boundary in case the boundary would be crossed. This is done mathematically by calculating all line-sphere intersections. The equation for a sphere and a line are combined and then we solve for the distance d. The equations for the sphere and the line are:

$$\text{Sphere:} \quad |\vec{x} - \vec{c}|^2 = r^2 \qquad \text{Line:} \quad \vec{x} = \vec{o} + d\vec{l}, \tag{2.19}$$

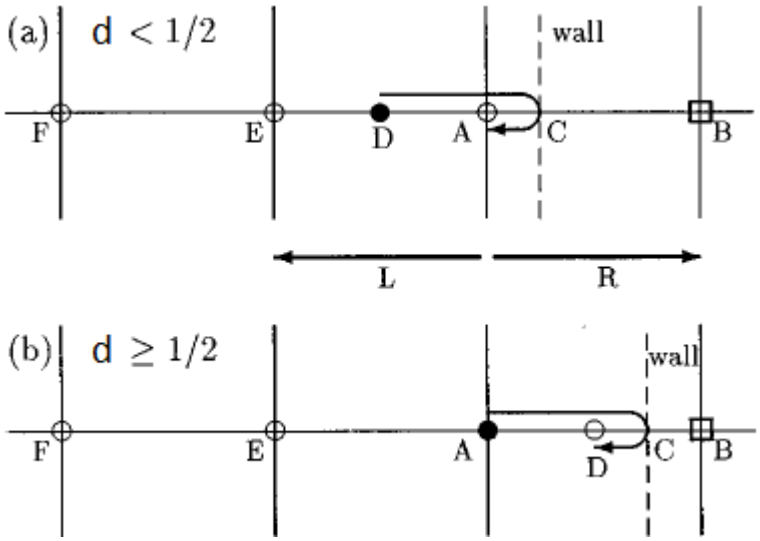


Figure 2.4: A schematic overview of the boundary locations used in the interpolation boundary conditions. In (a) the wall is closer to the fluid node than to the solid node, so $d < \frac{1}{2}\Delta x$. In (b) the wall is midway or closer to the solid node, here $d \geq \frac{1}{2}\Delta x$. Source:[4].

where $\vec{x}$ represents the points on the sphere or line respectively, r represents the radius of the sphere, $\vec{c}$ is the center of the sphere, $\vec{o}$ is the origin of the line (the fluid node on which the line starts) and $\vec{I}$ is the direction of the line (a unit vector). We now fill the line equation in into the sphere equation:

$$|\vec{o} + d\vec{I} - \vec{c}|^2 = r^2 \tag{2.20a}$$

$$\left(d\vec{I} + \vec{o} - \vec{c}\right) \cdot \left(d\vec{I} + \vec{o} - \vec{c}\right) = r^2 \tag{2.20b}$$

$$d^2\left(\vec{I}\cdot\vec{I}\right) + 2d\left(\vec{I}\cdot(\vec{o}-\vec{c})\right) + (\vec{o}-\vec{c})\cdot(\vec{o}-\vec{c}) = r^2 \tag{2.20c}$$

$$d^2\left(\vec{I}\cdot\vec{I}\right) + 2d\left(\vec{I}\cdot(\vec{o}-\vec{c})\right) + (\vec{o}-\vec{c})\cdot(\vec{o}-\vec{c}) - r^2 = 0. \tag{2.20d}$$

In equation (2.20d), we now see a second-degree polynomial arise, which we can solve with the standard quadratic formula:

$$d = \frac{-2\left(\vec{I}\cdot(\vec{o}-\vec{c})\right) \pm \sqrt{\left(\vec{I}\cdot(\vec{o}-\vec{c})\right)^2 - \vec{I}^2\left((\vec{o}-\vec{c})\right)^2 - r^2}}{\vec{I}^2} \tag{2.21a}$$

$$d = -\left(\vec{I}\cdot(\vec{o}-\vec{c})\right) \pm \sqrt{\left(\vec{I}\cdot(\vec{o}-\vec{c})\right)^2 - \left((\vec{o}-\vec{c})\right)^2 - r^2}. \tag{2.21b}$$

If the value under the square-root in equation (2.21b) is greater than zero, two solutions exist, which means the line touches the sphere in two points, and thus goes through the sphere. When the value under the root is zero the line just touches the sphere in one point and if the value under the root is less than zero the line does not intersect the sphere. Since we only need to know the distances from a fluid node to the sphere wall when a fluid node is next to a solid node, we only need distances which are smaller than the distance between to neighboring nodes, which is Δx. This is how we select the relevant distances d for the interpolation. Further we divide these distances in being greater or smaller than $\frac{1}{2}\Delta x$.

LINEAR INTERPOLATION BOUNDARY CONDITIONS

The linear interpolation formula we use from Bouzidi et al. [4] uses two nodes, namely the fluid boundary node $\vec{r}_j$ and the node the fluid population came from $\vec{r}_j - \vec{e}_q\Delta t$. In the case of figure 2.4 we use nodes A and E. The equations are

$$n_{q'}(\vec{r}_j, t+\Delta t) = 2d\, n_q^*(\vec{r}_j, t) + (1-2d)\, n_q^*(\vec{r}_j - \vec{e}_q\Delta t, t), \tag{2.22a}$$

$$n_{q'}(\vec{r}_j, t+\Delta t) = \frac{1}{2d} n_q^*(\vec{r}_j, t) + \frac{2d-1}{2d} n_{q'}^*(\vec{r}_j, t), \tag{2.22b}$$

where equation (2.22a) is for $d < \frac{1}{2}\Delta x$ and equation (2.22b) is for $d \geq \frac{1}{2}\Delta x$.

QUADRATIC INTERPOLATION BOUNDARY CONDITIONS

The quadratic interpolation formulas used are also from Bouzidi et al. [4] and here three nodes are used. The same nodes as with the linear interpolation boundaries and an additional neighboring node $\vec{r}_j - 2\vec{e}_q\Delta t$. In the case of figure 2.4, we here use nodes A, E and F. The formulas are

$$n_{q'}(\vec{r}_j, t+\Delta t) = d\,(2d+1)\,n_q^*(\vec{r}_j, t) + \left(1-4d^2\right) n_q^*(\vec{r}_j - \vec{e}_q\Delta t, t) - d\,(1-2d)\,n_q^*(\vec{r}_j - 2\vec{e}_q\Delta t, t), \tag{2.23a}$$

$$n_{q'}(\vec{r}_j, t+\Delta t) = \frac{1}{d\,(2d+1)} n_q^*(\vec{r}_j, t) + \frac{2d-1}{d} n_{q'}^*(\vec{r}_j, t) + \frac{1-2d}{1+2d} n_{q'}^*(\vec{r}_j - \vec{e}_q\Delta t, t), \tag{2.23b}$$

where equation (2.23a) is for $d < \frac{1}{2}\Delta x$ and equation (2.23b) is for $d \geq \frac{1}{2}\Delta x$.

MULTIREFLECTION BOUNDARY CONDITIONS

The linear and quadratic interpolation schemes both lead to second-order accuracy at boundaries [22], but the location depends on the viscosity in an unphysical way that cannot easily be eliminated. The multireflection boundary conditions are designed as third-order kinetic accurate boundary conditions for general flows. It uses more general relations to construct a boundary condition that is viscosity independent [8]. The boundary condition is derived from the Chapman-Enskog expansion at a planar boundary. The closure relation is as follows

$$\begin{aligned} n_{q'}(\vec{r}_j, t+\Delta t) = n_q(\vec{r}_j + \vec{e}_q\Delta t, t+\Delta t) + \frac{1-2d-2d^2}{(1+d)^2} n_q(\vec{r}_j, t+\Delta t) + \frac{d^2}{(1+d)^2} n_q(\vec{r}_j - \vec{e}_q\Delta t, t+\Delta t) \\ - \frac{1-2d-2d^2}{(1+d)^2} n_{q'}(\vec{r}_j - \vec{e}_q\Delta t, t+\Delta t) - \frac{d^2}{(1+d)^2} n_{q'}(\vec{r}_j - 2\vec{e}_q\Delta t, t+\Delta t). \end{aligned} \tag{2.24}$$

In figure 2.4 the point $\vec{r}_j + \vec{e}_q\Delta t$ is point B. Using equation (2.2), this relation (2.24) can, just as the linear and quadratic equations, be written in terms of the post-collision distributions:

$$\begin{aligned} n_{q'}(\vec{r}_j, t+\Delta t) = n_q^*(\vec{r}_j, t) + \frac{1-2d-2d^2}{(1+d)^2} n_q^*(\vec{r}_j - \vec{e}_q\Delta t, t) + \frac{d^2}{(1+d)^2} n_q^*(\vec{r}_j - 2\vec{e}_q\Delta t, t) \\ - \frac{1-2d-2d^2}{(1+d)^2} n_{q'}^*(\vec{r}_j, t) - \frac{d^2}{(1+d)^2} n_{q'}^*(\vec{r}_j - \vec{e}_q\Delta t, t). \end{aligned} \tag{2.25}$$

3

LBM MODELS APPLIED

In this chapter, we will explain the models we used and give the corresponding weighing factors. Further we give an extensive overview of the numerical models created and the implementation of the boundary conditions in the model. The notation for the grids we used is as follows: DxQy. Here x is the number of spatial dimensions and y is the total number of velocities q. We build a D2Q9 and a D3Q19 model. We chose to have 9 velocities in the two-dimensional model so that the geometry would be a rectangular grid, also the D2Q9 model is very commonly used and more accurate than the D2Q7 model according to Skordos [5]. With the D3Q19 model we have a square grid which is commonly used. We start by making a simple two-dimensional model with half-way bounce-back and periodic walls on all sides, then we make the same kind of model in three dimensions. After that we create the numerical formula for solving the distances d needed for the implementation of the interpolation boundary techniques. Finally, we introduce linear interpolation, quadratic interpolation and multireflection boundary conditions using that numerical formula.

3.1. THE D2Q9 MODEL WITH HALF-WAY BOUNCE-BACK

The discrete velocity directions for the D2Q9 lattice are depicted in figure 3.1. The lattice-Boltzmann method gives us the Navier-Stokes equation with higher-order terms, but as long as the divergence $(\nabla \cdot \rho \vec{u})$ is kept small, the compressible Navier-Stokes equation is recovered, as can be seen in equation (2.1). Therefore the velocities must be smaller than the sound speed ($Ma \equiv \frac{u}{c_s} \ll 1$). The weighting factors ω_q for the D2Q9 grid and the sound speed are shown in table 3.1. Here rest refers to a particle with $\vec{e}_9 = 0$, slow refers to a lattice velocity pointing in only one spatial direction (x, y or z) and fast refers to a lattice velocity which points in at least two spatial directions. Now combining figure 3.1 and table 3.1, we see that e_9 is the rest particle, e_1, e_3, e_5 and e_7 are slow and e_2, e_4, e_6 and e_8 are fast particles.

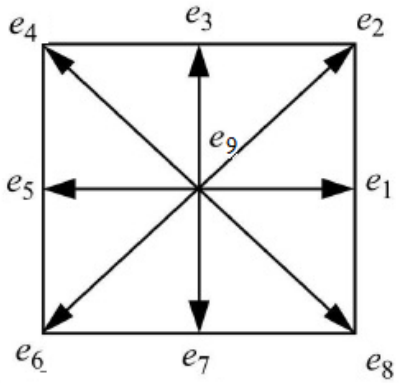


Figure 3.1: The D2Q9 grid model used for building the LBM algorithm.

The particle distribution functions at each lattice point are updated using equation (2.16). This equation holds for all lattice points within the fluid, but not for the domain boundaries. Here boundary conditions compensate for the insufficient number of particle distribution functions. For this reason, the streaming

Grid Model	ω_q(rest)	ω_q(slow)	ω_q(fast)	c_s^2
D2Q9	$\frac{4}{9}$	$\frac{1}{9}$	$\frac{1}{36}$	$\frac{1}{3}$

Table 3.1: The grid setup for the D2Q9 model. Rest refers to a particle with $\vec{e}_9 = 0$, slow refers to a lattice velocity pointing in only one spatial direction (x, y or z) and fast refers to a lattice velocity which points in at least two spatial directions. Source: [17]

step, the collision step and the body force step should always be treated separately in numerical implementations.

Building a D2Q9 model is done as follows:

1. Define velocity vectors for the model, as in figure 3.1.
2. Define the gridsize (in our case x = 1:xmax ls by y = 1:ymax ls) and solid obstacles. The solid object used here is a circle (or cilinder) in the middle of the grid.
3. Initialize the density (as in table 3.1) and the probability distribution functions.

Now start the while loop.

4. Compute the macroscopic density and velocity, as in equation (2.5).
5. Compute the equilibrium distribution function, as in equation (2.14).
6. Now implement the collision step, as in formula (2.11).
7. Add the body force to the distribution function, as done in equation (2.16).
8. Initialize the bounce-back boundaries, in this case half-way bounce-back as in equation (2.18).
9. Implement the streaming step as in equation (2.2) and the periodic boundaries. Periodic boundaries can be created by adding extra cells, so-called 'ghost' nodes, at the end of the domain. Allocate an array of size (xmax+2,ymax+2) and call the last element in this array (-1,-1) instead of (0,0). This gives access to points that are one place outside the domain, and these cells can be filled. We can set (-1,y) = (xmax,y) for example.
10. Implement the bounce-back boundaries.

End the while loop here. The iteration continues until convergence is reached.

The D2Q9 MATLAB code build can be found in appendix A.1.

3.2. THE D3Q19 MODEL

The discrete velocity directions for the D3Q19 lattice are depicted in figure 3.2. The weighting factors ω_q for the D3Q19 grid model and the sound speed are shown in table 3.2. When combining figure 3.2 and table 3.2, we see that e_1 is the rest particle here.

Grid Model	ω_q(rest)	ω_q(slow)	ω_q(fast)	c_s^2
D3Q19	$\frac{1}{3}$	$\frac{1}{18}$	$\frac{1}{36}$	$\frac{1}{3}$

Table 3.2: The grid setup for the D3Q19 model. Rest refers to a particle with $\vec{u} = 0$, slow refers to a lattice velocity pointing in only one spatial direction (x, y or z) and fast refers to a lattice velocity which points in at least two spatial directions. Source: [17]

Building a D3Q19 model with half-way bounce-back is done in the same way as the D2Q9 model with half-way bounce-back. Only here there of course is a third dimension (z) and there are 19 velocity directions instead of 9. To build this model exactly the same steps as used for building the D2Q9 model above can be used. The D3Q19 half-way bounce-back MATLAB code can be found in appendix A.2.

To be able to implement the interpolation boundary conditions, we first need to calculate the distances from all grid nodes to the sphere. We first determine all cases in which the solid boundary is crossed. This happens when we have a fluid node $\vec{r}_j$ such that $\vec{r}_j + \vec{e}_q \Delta t$ is a solid node. Since the geometries researched

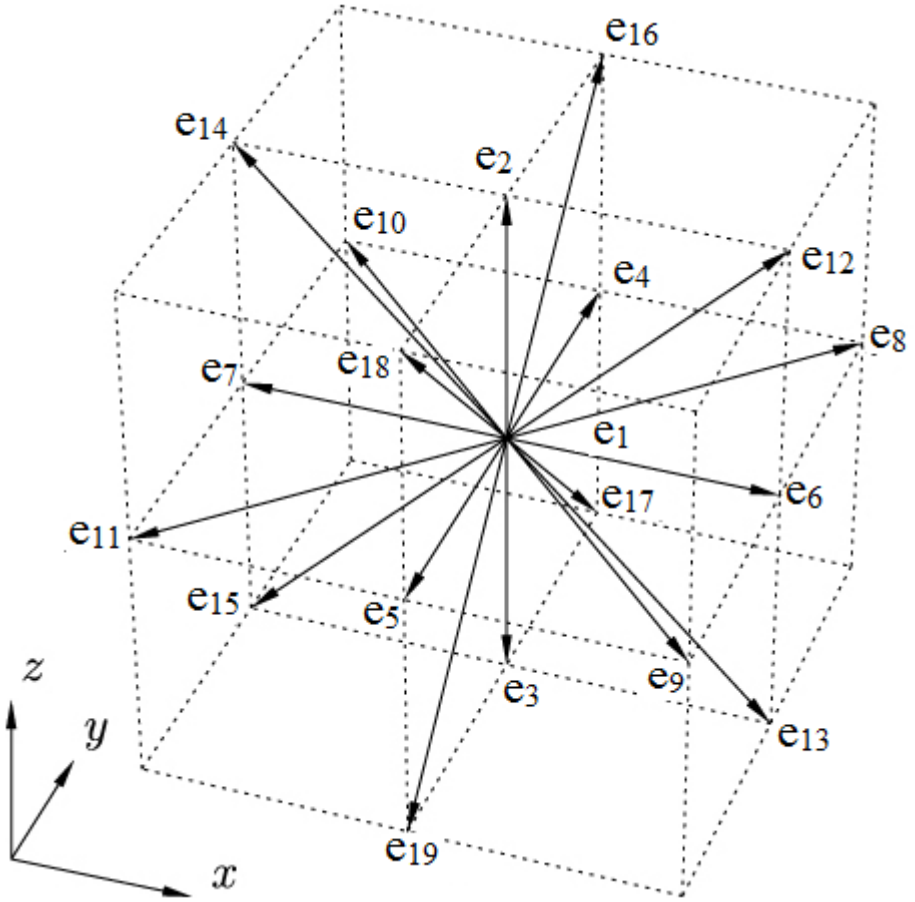


Figure 3.2: The D3Q19 grid model used for building the LBM algorithm.

in this thesis are spheres we search for all line-sphere intersections. As is derived in section 2.2.3 the formula for the distances is equation (2.21b). We use this formula to calculate the distances to the sphere of each grid node. We only use the distances for the fluid populations that are on grid nodes from which they can cross the solid boundary. For calculating where these fluid populations will end up in the next time step we use the interpolation formulas. For this we need to know the location of this fluid population now and one (and two) timesteps ago. These locations are also calculated in our geometric code. The geometric formula we build for calculating the line-sphere intersections and the locations and velocity directions of our fluid populations can be found in appendix A.3.

When we now want to build our D3Q19 models with interpolation boundary conditions, all we need to do is call the line-sphere intersection formula somewhere before the time loop so we have our distances and then we implement our specific boundary conditions instead of the half-way bounce-back ones. We use equations (2.22a) and (2.22b) for the linear interpolation boundary conditions, equations (2.23a) and (2.23b) for the quadratic interpolation boundary conditions and equation (2.25) for the multireflection boundaries. Our D3Q19 linear interpolation, quadratic interpolation and multireflection boundary condition MATLAB codes can be found in appendix A.4, A.5 and A.6, respectively.

4

PHYSICAL CONSISTENCY

Simulations at Reynolds = 0.5 were performed, in order to compare our results with physical reality. The benchmark we used is an analytical solution of the drag force for a dilute cubic array of spheres at Stokes flow, derived by Hasimoto [23]. The benchmark is valid for a periodic array of spheres under steady-state conditions, at Stokes flow (here Reynolds = 0.5). In this case our grids are defined as in figure 4.1.

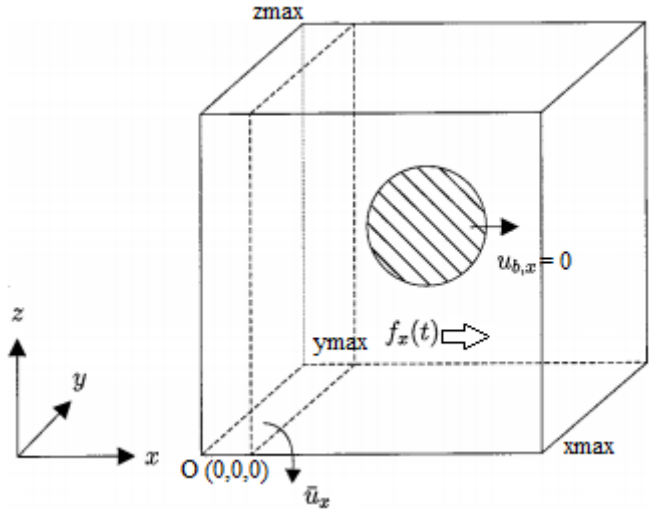


Figure 4.1: Fluid flowing around a solid sphere (at $u_{b,x} = 0$) in a fully periodic computational domain. A time-dependent body force, causes a fluid flow in the x-direction. Source:[24]

The analytical drag force on a cubic array of spheres with diameter D and fluid velocity $\vec{u}_x$ in a domain of size L^3 and with a boundary velocity $\vec{u}_{b,x} = 0$ at every boundary point, reads [23]

$$\vec{F}_{d,Has} = \frac{3\pi\rho\nu D\bar{u}_x}{1 - 1.7601\sqrt[3]{\phi} + \phi - 1.5593\phi^2}, \tag{4.1}$$

where $\phi = \frac{\pi D^3}{6L^3}$ represents the volume fraction of spheres in the array. When $\phi \to 0$, the drag force would go to $\vec{F}_{d,Has} = 3\pi\rho\nu D\bar{u}_x$. $\phi \to 0$ is the case where only one sphere is left, and for this case the drag coefficient $C_d = \frac{24}{Re}$, for $Re < 1$. The drag equation gives us the drag force as $\vec{F}_d = \frac{1}{2}\rho\vec{u}^2 C_d A$. When filling in the C_d for one sphere and the frontal area of the sphere $A = \frac{\pi D^2}{4}$, we find that $\vec{F}_d = \frac{3}{Re}\rho\vec{u}^2\pi D^2$. Since $\frac{D\vec{u}}{Re} = \nu$, we here find our analytical solution in the case of one sphere: $\vec{F}_d = 3\pi\rho\nu D\vec{u}$.

We have implemented the analytical solution in our code, to see how accurately our different boundary techniques could find the corresponding Reynolds number. This is done as follows. First, we calculated the total volume of the fluid, which is the total volume minus the volume of the sphere. Now the friction force on or sphere would be

$$\vec{F} = \rho V_f \vec{g}, \tag{4.2}$$

Dependent variable	Input variable	Kept constant
Reynolds	ϕ	Diameter of sphere (D)
Reynolds	$\frac{\Delta x}{D}$	ϕ (and therefore also ($\frac{D}{L}$)
Reynolds	ϕ	D per boundary technique used

Table 4.1: The simulations done in this thesis. In each case one variable is changed (the input variable) and other variables are kept constant to see the influence of the input variable on the Reynolds number.

where ρ is the density, V_f is the volume of the fluid and $\vec{g}$ is the magnitude of acceleration. In this case the friction force should be equal to the analytically obtained drag force on the sphere from equation (4.1) because this is the only other force in the x-direction here and the sphere does not move from its position. In the y-direction the gravitation force and the normal force also compensate. When equating the friction and the drag force we obtain an expression for the magnitude of acceleration, namely

$$\vec{g} = \frac{3\pi\nu D\bar{u}_x}{V_f(1-1.7601\sqrt[3]{\phi}+\phi-1.5593\phi^2)}. \tag{4.3}$$

When we now use that $Re = \frac{D\bar{u}_x}{\nu}$, we find that

$$\vec{g} = \frac{3\pi\nu^2 Re}{V_f(1-1.7601\sqrt[3]{\phi}+\phi-1.5593\phi^2)}. \tag{4.4}$$

For our simulations, spheres with 3 different diameters D = 6,8 and 12 ls were used and grid sizes in the range from L = 12 ls to L = 128 ls were used. We have collected results for the cases given in table 4.1.

The viscosity used in our simulations is $\nu = \frac{1}{6} ls^2 \cdot lt^{-1}$, because it was found that at this viscosity the smallest deviations between de measured drag force and the analytical drag force occur when using our bounce-back principle [25].

We have implemented the magnitude of acceleration and used it to calculate the body force. This body force is applied each time step, and our iteration stops when convergence of the calculated Reynolds number is found. We compare the Reynolds numbers found by our simulations with the implemented Reynolds = 0.5 in the analytical formula in the results section.

5

RESULTS AND DISCUSSION

In this chapter several results from our simulations will be presented. The different boundary condition techniques will be compared in accuracy and computation time. First we will compare the boundary techniques for different sphere refinements, than for different ϕ, than we will look at sphere refinement effects in all boundary conditions separately, and finally we will compare the computational efficiency of the boundary techniques. The calculations were seen as converged when the error in the Reynolds number was smaller than 0.000001. The error in the Reynolds number is defined as

$$\varepsilon = \frac{Re(t) - Re(t-1)}{Re(t-1)}, \tag{5.1}$$

where ε is the error, Re (t) is the Reynolds number calculated in the latest time step (from the last fluid movement step) and Re (t-1) is the Reynolds number calculated from the previous time step.

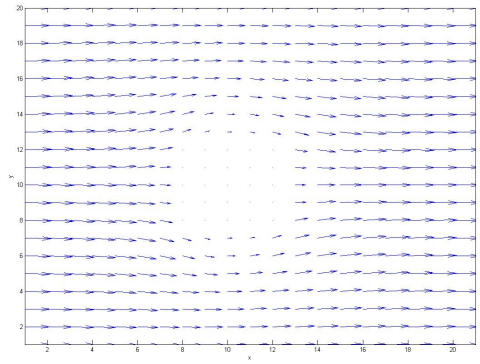
5.1. FLOW PATTERNS

To be able to see how accurately flow around a sphere is portrayed for each boundary technique we have made flow patterns. These are flow patterns on a grid with size 20 ls by 20 ls by 20 ls and a sphere with a diameter of 6 ls. We give a two-dimensional representation of the x- and y-axis, where z = 10 ls. In figure 5.1 the flow patterns are depicted for all boundary condition techniques. As we can see the sphere seems to be represented by a cube when using half-way bounce-back boundaries as in figure 5.1(a). Qualitatively we cannot see a clear difference between the linear and quadratic interpolation boundary techniques in figures 5.1(b) and 5.1(c), respectively, but these techniques are both more accurate than the half-way bounce-back technique. The multireflection boundary technique in figure 5.1(d) gives a little more refined flow pattern than the other interpolation techniques, as the flow arrows directly around the sphere are smaller than with the linear and quadratic interpolation technique.

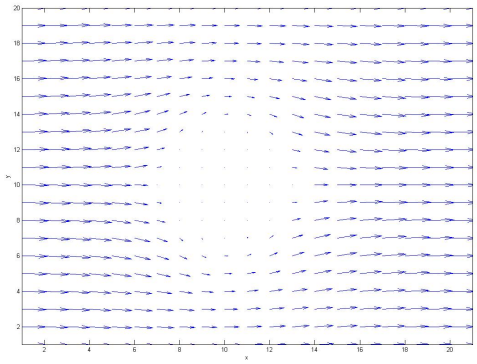
5.2. BOUNDARY TECHNIQUES COMPARED AT DIFFERENT SPHERE REFINEMENTS

In our experiments we have used spheres with a diameter of 6, 8 and 12 grid points (ls) on grids varying from 12 by 12 by 12 to 128 by 128 by 128 grid points (ls). The Reynolds number is calculated as $Re = \frac{D\bar{u}_x}{\nu}$, where D is the diameter of the sphere, ν is the viscosity of the fluid and $\bar{u}_x$ is the average fluid velocity in the x-direction. $\bar{u}_x$ is calculated as $\bar{u}_x \equiv \frac{\sum u_x(nx+r+2ls,:,:)}{N}$, which means that the fluid velocity is measured 2 ls after the end of the sphere in the x-direction (nx represents the center of the sphere and r the radius of the sphere) for each case. At this point in the x-direction, we have taken all nodes in the y- and z-direction. So we have taken the whole plane in the total flow field at 2 ls behind the sphere in the x-direction. We have calculated the average value of the fluid velocity in this plane.

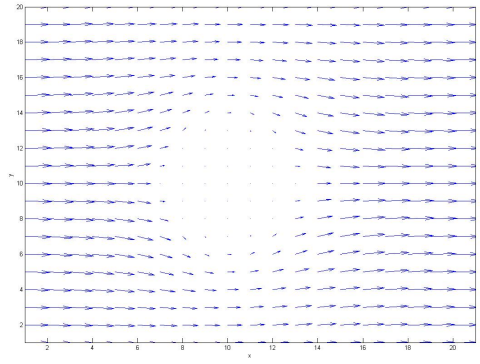
In figure 5.2 the deviation of the simulated Reynolds number from the benchmark is shown as a function of ϕ, the volume fraction of spheres in the array, for a sphere with a diameter of 6 ls. This is the coarsest sphere we have simulated. As we can see in the left figure the half-way bounce-back boundary technique has the largest error at coarse grids with few grid points but the error decreases to about the same error as the interpolation techniques at more refined grids. This is as we expected it because on coarse grids the difference



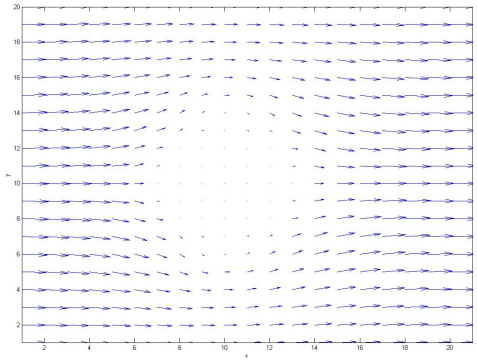
(a) Half-way bounce-back



(b) Linear interpolation



(c) Quadratic interpolation



(d) Multireflection

Figure 5.1: Two-dimensional representation of a lattice-Boltzmann calculation of a flow past a sphere when using the half-way bounce-back, linear interpolation, quadratic interpolation or multireflection boundary technique, respectively. The arrows represent the direction and magnitude of flow, which is forced from the left to the right. This representation is made with a sphere with a diameter of 6 ls on a grid of size 20 ls by 20 ls by 20 ls. The pattern is taken where z = 10 ls.

in accuracy between the half-way bounce-back and the interpolation techniques is more pronounced than on finer grids. The linear and quadratic interpolation technique do not differ a lot, we can see that the linear interpolation technique is a little closer to the benchmark. The multireflection boundary technique gives us strange results. The error in the Reynolds number is quite constant everywhere, and we would expect that the error would decrease as ϕ goes to zero. We expect this because as ϕ goes to zero, the influence of the sphere decreases. The average fluid velocity is less dependent on the sphere influences because many nodes are taken into account. We therefore think that there is a mistake in the implementation of this technique because the simulated Reynolds number is about the same for every ϕ. Something that is strange about all boundary techniques is that the simulated Reynolds numbers for all techniques have values underneath the benchmarked Reynolds number for small ϕ. We think that there is a constant error in our simulations which is negative, so in fact our deviations from the benchmarked Reynolds number are a little larger. In figure 5.2 on the right we have depicted the same data but now the y-axis is also logarithmic. We have done this to be able to see the order of the error in the Reynolds number. Due to the fact that the error becomes negative when ϕ gets smaller, and this cannot be displayed in a logarithmic plot, we have taken the absolute value of the error. This is why at a certain point the error seems to get larger again, but actually it is just negative. When we look at the slope of the positive errors, we can see that the half-way bounce-back technique has a steeper slope $\left(\frac{d\varepsilon}{d\phi}\right)$ than the linear and quadratic interpolation techniques, indicating that the half-way bounce-back technique has an error that decreases faster when ϕ decreases. This is not what we expected, because the interpolation techniques are much more refined and are second order accurate according to other papers. The interpolation procedure should work better when there are more nodes.

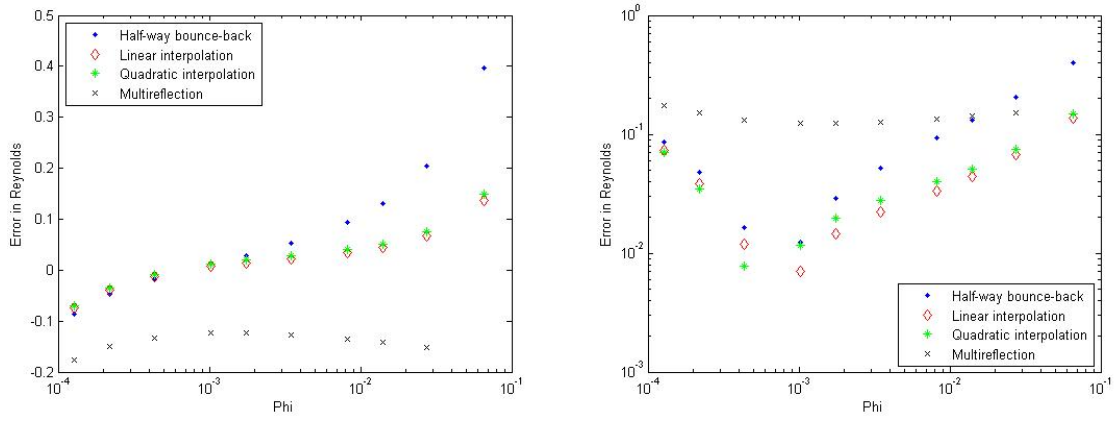


Figure 5.2: The simulated error in the Reynolds number versus phi, for all boundary techniques applied to a sphere with D = 6 ls. The analytical Reynolds number is set to 0.5. In the graph on the right the logarithm of the error is taken to be able to see the order of the error of the simulated Reynolds number compared to the analytical Reynolds number.

In figure 5.3 the deviation of the simulated Reynolds number from the benchmark is shown as a function of ϕ, the volume fraction of spheres in the array, for a sphere with a diameter of 8 ls. This is the middle sphere we have simulated. As we can see a lot is alike between the spheres with a diameter of 6 and 8 ls. The half-way bounce-back boundary technique again has the largest error at coarse grids with few grid points but the error decreases to about the same error as the interpolation techniques at more refined grids, as we expected. The linear and quadratic interpolation technique do not differ a lot, we can see that the linear interpolation technique is a little closer to the benchmark. The multireflection boundary technique again gives us strange results, confirming that we have made an implementation error. The error in the Reynolds number is quite constant everywhere, and at some points even increases when the volume fraction of spheres gets smaller. In figure 5.3 on the right we have depicted the same data but now the y-axis is also logarithmic. Again, due to the fact that the error becomes negative when ϕ gets smaller, and this cannot be displayed in a logarithmic plot, we have taken the absolute value of the error. This is why at a certain point the error seems to get larger again, but actually it is just negative. When we look at the slope $\left(\frac{d\varepsilon}{d\phi}\right)$ of the positive errors, we can see that the half-way bounce-back technique again has a steeper slope than the linear and quadratic interpolation techniques, indicating that the error in the Reynolds number of the half-way bounce-back technique decreases faster when ϕ gets smaller than the error of the interpolation techniques. This is as with the coarser sphere not what we expected.

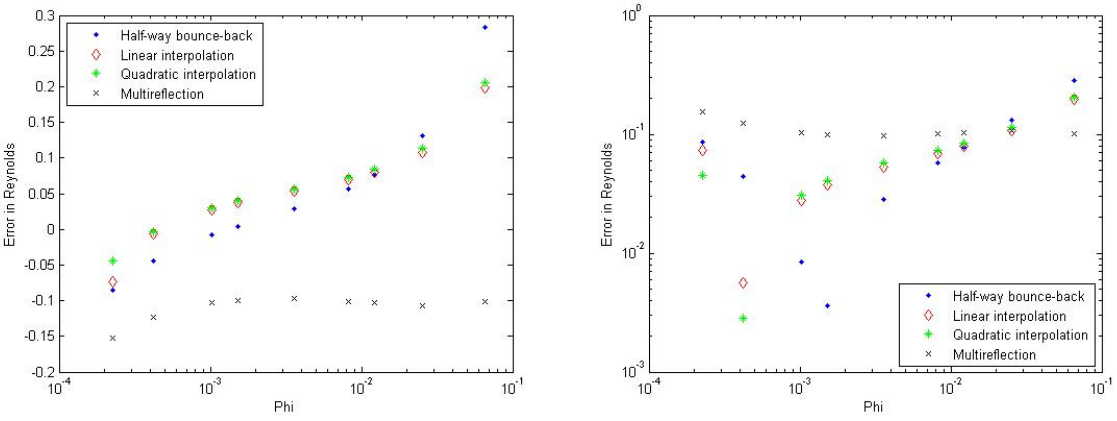


Figure 5.3: The simulated error in the Reynolds number versus phi, for all boundary techniques applied to a sphere with D = 8 ls. The analytical Reynolds number is set to 0.5. In the graph on the right the logarithm of the error is taken to be able to see the order of the error of the simulated Reynolds number compared to the analytical Reynolds number.

In figure 5.4 the deviation of the simulated Reynolds number from the benchmark is shown as a function of ϕ, the volume fraction of spheres in the array, for a sphere with a diameter of 12 ls. This is the most re-

fined sphere we have simulated. As we can see the linear and quadratic interpolation boundary techniques give similar results as with the spheres with a diameter of 6 and 8 ls. The half-way bounce-back boundary technique here gives us only negative results, which indicates a simulated Reynolds number which lies below the benchmark. We think that there is a constant error in our simulations which is negative, but it is very strange that it is different from the interpolation techniques only for this sphere diameter. The multireflection boundary technique again gives us strange results, confirming that we have made an implementation error. In figure 5.4 on the right we have depicted the same data but now the y-axis is also logarithmic. Again, due to the fact that the error becomes negative when ϕ gets smaller, and this cannot be displayed in a logarithmic plot, we have taken the absolute value of the error. This is why at a certain point the error seems to get larger again, but actually it is just negative. The half-way bounce-back and multireflection boundary technique both seem to have a negative slope $\left(\frac{d\varepsilon}{d\phi}\right)$ but this is due to the negative errors. In fact they have a same size positive slope. Here the slope of the interpolation techniques is steeper than the slope of the half-way bounce-back technique indicating that the errors in those techniques decrease faster when ϕ gets smaller, as we would expect it.

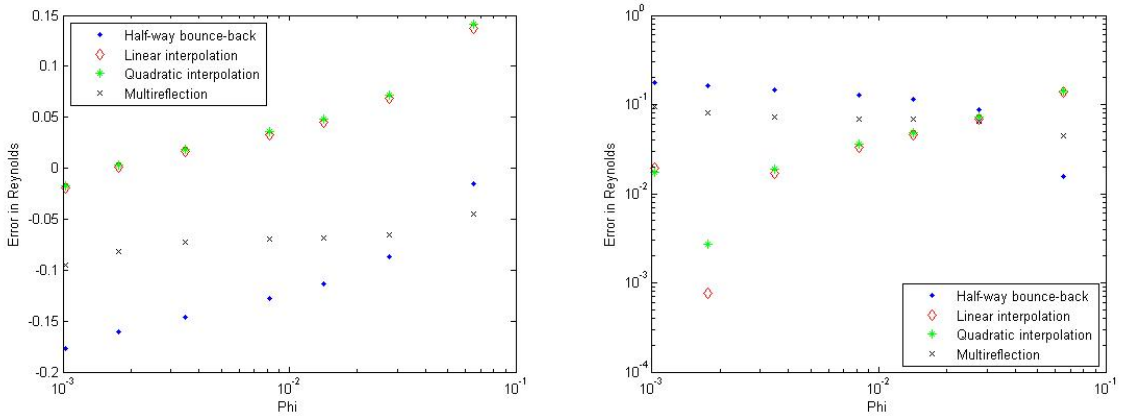


Figure 5.4: The simulated error in the Reynolds number versus phi, for all boundary techniques applied to a sphere with D = 12 ls. The analytical Reynolds number is set to 0.5. In the graph on the right the logarithm of the error is taken to be able to see the order of the error of the simulated Reynolds number compared to the analytical Reynolds number.

5.3. BOUNDARY TECHNIQUES COMPARED AT DIFFERENT ϕ

In figures 5.5, 5.6 and 5.7 we have (logarithmically) plotted the error in Reynolds versus $\frac{\Delta x}{D}$, the refinement of the sphere. As we can see for the linear and quadratic interpolation techniques the error is about the same for the coarse and refined sphere, but is larger for the middle sphere. We would have expected that the error would in all cases be the largest for the coarsest sphere (D = 6 ls) and the smallest for the most refined sphere (D = 12 ls), but this is not the case. For the half-way bounce-back technique, in figure 5.5 where ϕ and so the volume fraction of spheres in the array is the smallest the error is larger for the coarser sphere as we would expect, but this is again not the case for the middle sphere. For the most refined sphere the error is also larger for the coarser sphere, which can be seen in the right figure of figure 5.7. The error in the Reynolds number for the multireflection boundary technique is almost constant, but a little larger for better refined spheres. This again does not make sense. We think that there is a mistake in our MATLAB code, because when $\frac{\Delta x}{D} \to 0$, the error in the Reynolds number should always be smaller. A more refined sphere should always lead to a more accurate calculation of the Reynolds number. A semi-logarithmic plot (logarithmic in the vertical axis) should show a straight line with a positive slope. The slope is than gives the order of spatial accuracy. These kind of results have been found by others, for example by AhrenHolz et al. [26] (they used the resolution on the horizontal axis which is why their slope is negative instead of positive).

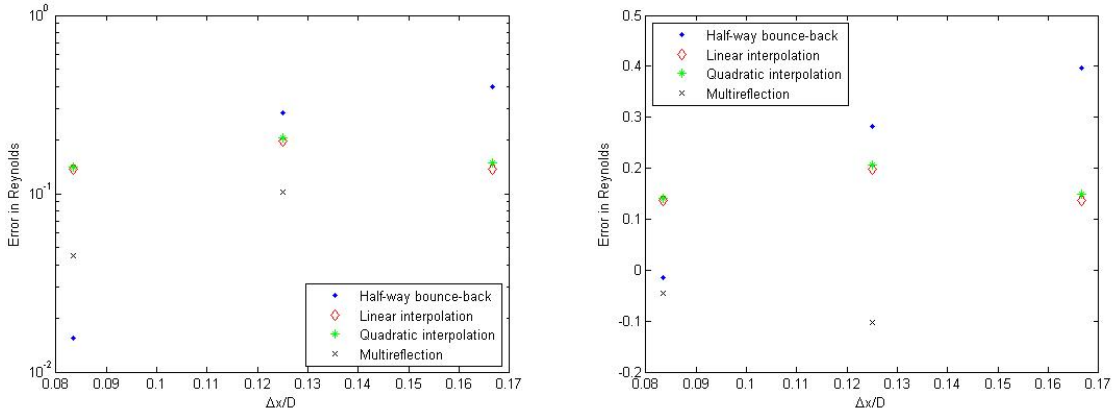


Figure 5.5: The simulated error in the Reynolds number versus $\frac{\Delta x}{D}$, for all boundary techniques and when keeping $\frac{D}{L}$, and therefore ϕ, constant at $\frac{D}{L} = \frac{1}{2}$. The analytical Reynolds number is set to 0.5. In the graph on the right the sign of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

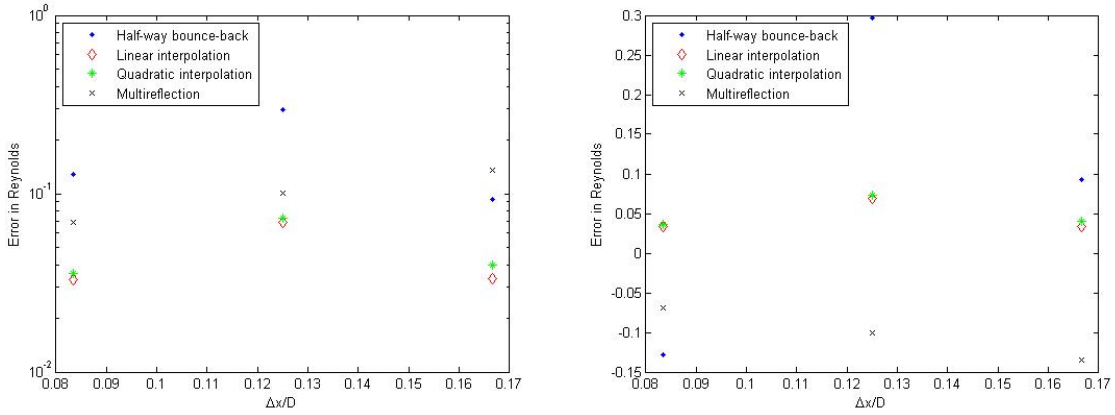


Figure 5.6: The simulated error in the Reynolds number versus $\frac{\Delta x}{D}$, for all boundary techniques and when keeping $\frac{D}{L}$, and therefore ϕ, constant at $\frac{D}{L} = \frac{1}{4}$. The analytical Reynolds number is set to 0.5. In the graph on the right the sign of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

5.4. SPHERE REFINEMENT EFFECTS FOR THE DIFFERENT BOUNDARY CONDITIONS

In the previous section we saw that for some techniques, at a constant ϕ, the error was the largest for the middle sphere. To study this further we look at figures 5.8, 5.9, 5.10 and 5.11. In figure 5.8 the (error in the) Reynolds number is plotted as a function of ϕ for the half-way bounce-back technique. When ϕ gets smaller the error should get smaller, because the average fluid velocity than depends less on the influence of the sphere. Here we see that when ϕ gets smaller the Reynolds number does get smaller, so the shape of the graphs for all three sphere refinements is as we would expect it, but the error does not always get smaller. This happens because in some cases the simulated Reynolds number is lower than the benchmarked Reynolds number. We think that there is a constant error in our simulations, and that when correcting for this error the simulated Reynolds number should be very close to 0.5 for all techniques when ϕ is very small. If this is the case than the constant error would be especially large for the most refined sphere (D = 12 ls) about 20 %. It would then also be the case that the error is the smallest for the most refined sphere and the largest for the coarsest sphere, which is as we would expect it.

In figure 5.9 and figure 5.10 the (error in the) Reynolds number is plotted as a function of ϕ for the linear and quadratic interpolation technique, respectively. As we can see here the middle sphere (D = 8 ls) has the largest error and the coarsest (D = 6 ls) and most refined sphere (D = 12 ls) are the same for coarse grids, but the coarsest sphere has a smaller error on more refined grids. The shape of the graphs for all three sphere

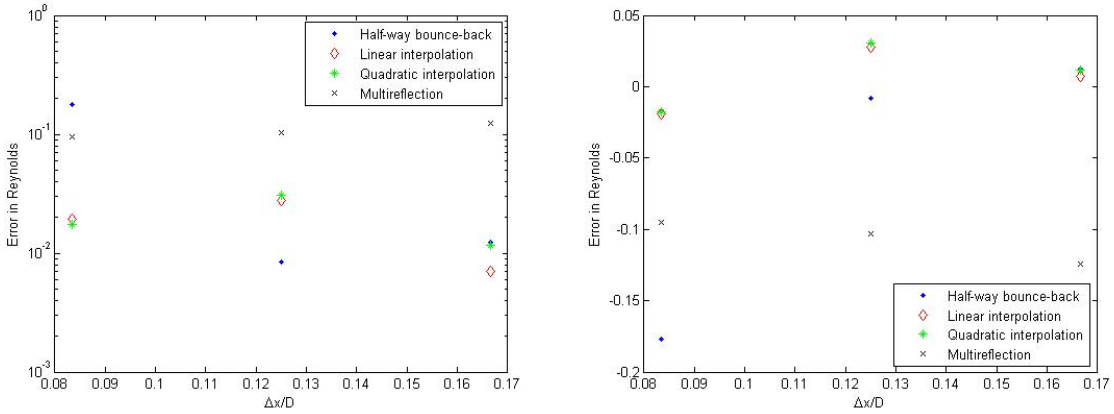


Figure 5.7: The simulated error in the Reynolds number versus $\frac{\Delta x}{D}$, for all boundary techniques and when keeping $\frac{D}{L}$, and therefore ϕ, constant at $\frac{D}{L} = \frac{1}{8}$. The analytical Reynolds number is set to 0.5. In the graph on the right the sign of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

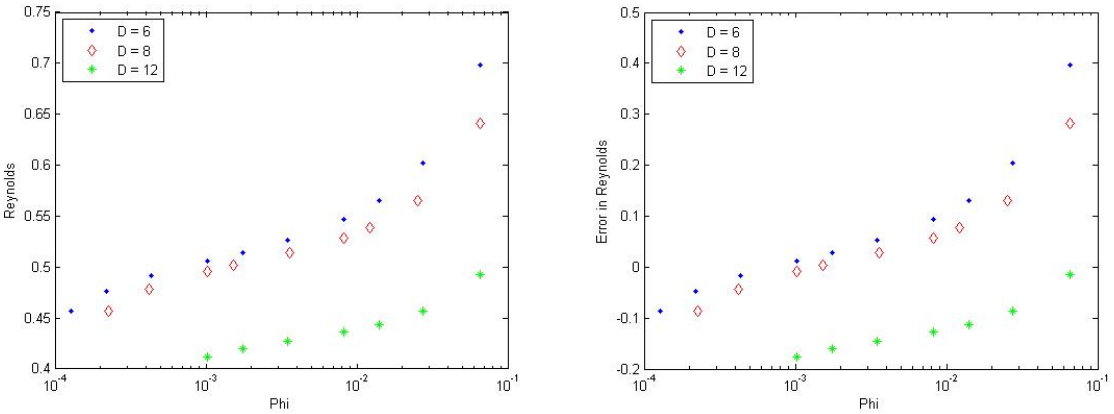


Figure 5.8: The simulated Reynolds number versus phi, for the half-way bounce-back technique applied to three different sphere sizes. The analytical Reynolds number is set to 0.5. In the graph on the right the size of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

refinements is as expected at high ϕ, but for larger grid sizes the Reynolds number seems to drop a lot, so when ϕ gets smaller than 10^{-3} the influence of the sphere seems to get bigger. This cannot be correct because the average fluid velocity (and therefore the Reyolds number) should be less dependent on the influence of the sphere when ϕ gets smaller. When $\phi \to 0$, we are left with only one sphere, which can only have limited influence compared to more spheres.

In figure 5.11 the (error in the) Reynolds number is plotted as a function of ϕ for the multireflection boundary technique. As we can see here the most refined sphere (D = 12 ls) has the smallest error and the coarsest (D = 6 ls) sphere has the largest error. This is what we would expect. The shape of the graphs for all three sphere refinements is as expected at high ϕ, but for larger grid sizes the Reynolds number seems to drop a lot and the influence of the sphere seems to get bigger. This cannot be correct because the average fluid velocity (and therefore the Reyolds number) should be less dependent on the influence of the sphere when ϕ gets smaller. This technique is implemented in the wrong way causing these strange results.

5.5. BOUNDARY TECHNIQUES COMPARED IN COMPUTATIONAL EFFICIENCY

When using the boundary techniques researched and implemented in this study the computation time is also important. To be able to compare the computation time of the different boundary techniques the wall clock time for each simulation was noted. In figure 5.12 the wall clock computation times for the three different sphere refinements and for all techniques are shown. As we can see in the figures the half-way bounce-back technique is the fastest method for all three sphere refinements. This is as we expected because it is

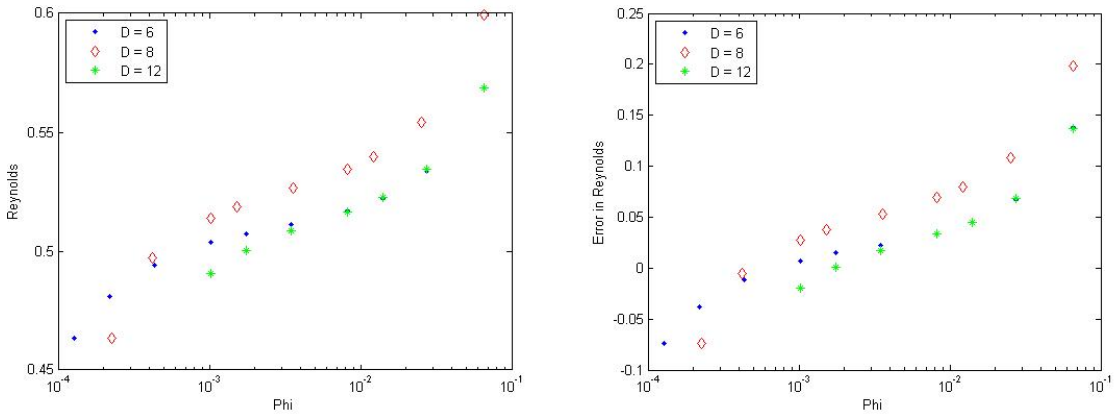


Figure 5.9: The simulated Reynolds number versus phi, for the linear interpolation boundary technique applied to three different sphere sizes. The analytical Reynolds number is set to 0.5. In the graph on the right the size of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

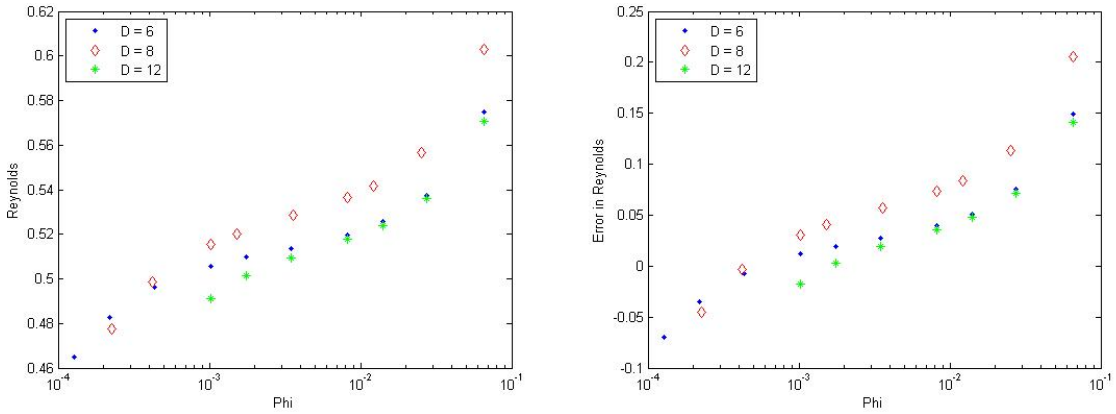


Figure 5.10: The simulated Reynolds number versus phi, for the quadratic interpolation boundary technique applied to three different sphere sizes. The analytical Reynolds number is set to 0.5. In the graph on the right the size of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.

the simplest method and it is does not require interpolation in the time loop. In figure 5.12(a) we can see that the quadratic interpolation technique has the longest computation time for very refined grids with the coarsest sphere. Further the linear and quadratic interpolation and multireflection technique show quite similar computation times. For the other two sphere refinements these three boundary techniques also show quite similar computation times.

What we can also see from figure 5.12 is that when the sphere is more refined the computation time is lower. For a sphere with D = 6 ls on a grid of $L^3 = 10^5 ls^3$ the computation time is about 30000 seconds and for a sphere with D = 12 ls on a grid of $L^3 = 10^5 ls^3$ 4000 seconds. This is very strange, because when simulating flow around a more refined sphere we would expect this to take longer than for a coarser sphere.

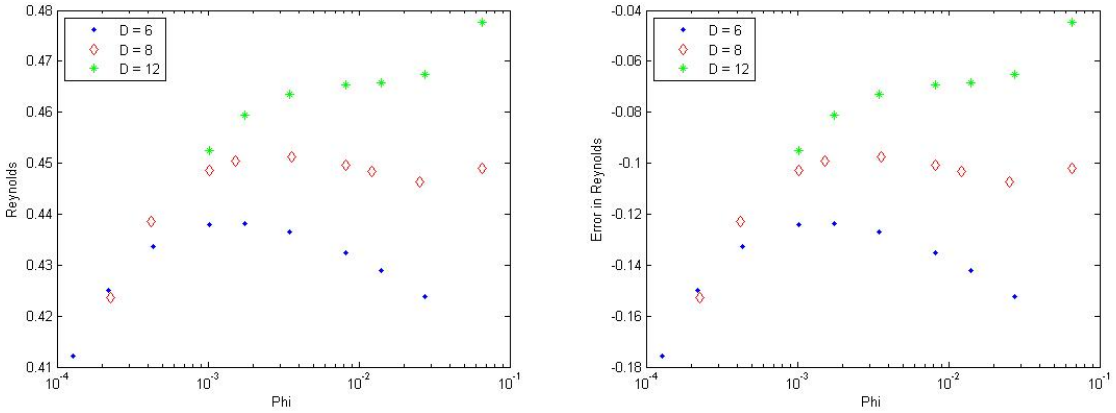
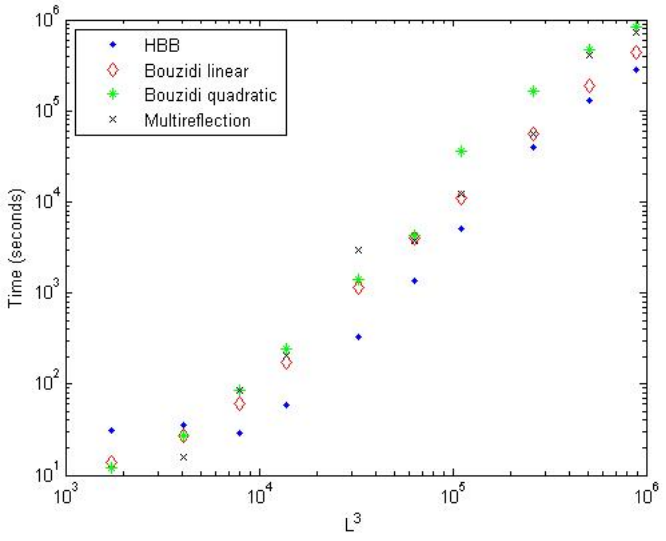
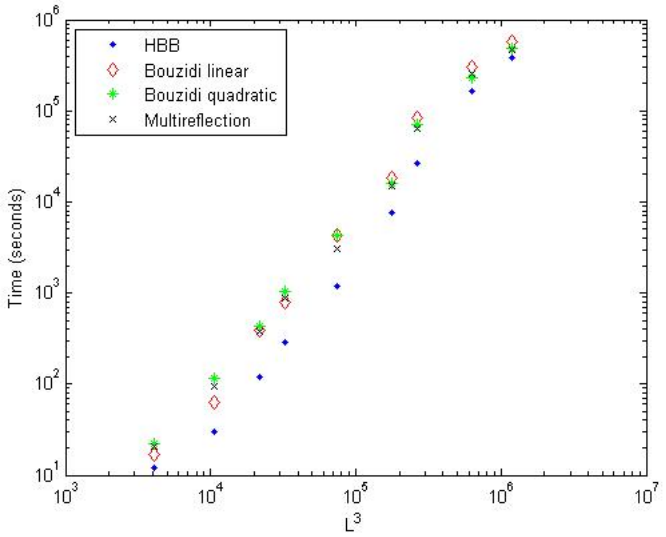


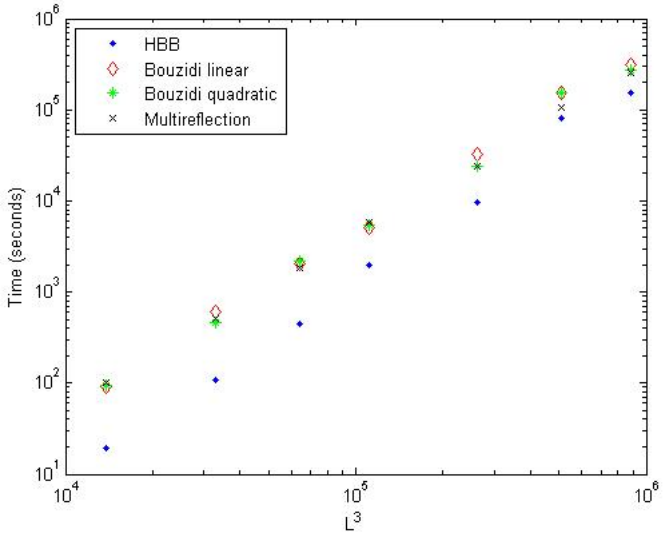
Figure 5.11: The simulated Reynolds number versus phi, for the multireflection boundary technique applied to three different sphere sizes. The analytical Reynolds number is set to 0.5. In the graph on the right the size of the error of the simulated Reynolds number compared to the analytical Reynolds number is made more clear.



(a) D = 6



(b) D = 8



(c) D = 12

Figure 5.12: The wall clock time of the simulation versus the grid size, for all boundary techniques applied to three different sphere refinements.

6

CONCLUSIONS AND RECOMMENDATIONS

The lattice-Boltzmann method with the half-way bounce-back boundary technique has proven to be a simple to implement and quite accurate method for simulating fluid flow around objects. However the implementation of interpolation boundaries was not as simple, and will be very hard for more complex shapes. To be able to implement those boundary conditions, a geometrical algorithm for calculating the distances between grid points and the outer points on the sphere had to be written. In this algorithm not only the distances are calculated, but also the fluid populations which will enter into the object in the next time step are selected and tracked for a number of points, depending on the accuracy of the implementation method, making this a complicated method. When having written this algorithm however it could be used for all interpolation methods.

When comparing the different boundary techniques it is striking that when using relatively fine grids the half-way bounce-back method is about as accurate as the interpolation techniques, when using more refined spheres this effect is even more clear. This is why when using these boundary conditions in practice we would use the half-way bounce-back method on a fine grid because of its easy implementation. When however computation time is of the essence, it could be good to simulate on a coarser grid because the computation time is much lower in that case. On a coarser grid the interpolation boundary techniques are more accurate than the half-way bounce-back technique. There is only a small difference in accuracy between the linear and quadratic interpolation techniques, even in favor of the linear interpolation technique, which is why we would recommend to use this technique in the case of coarse grids.

The multireflection boundary technique was not implemented in the right way in this study. For future research into this technique we would recommend looking at the interpolation behavior. For the other two interpolation techniques the distances from grid points to the sphere were separated in two categories: larger and smaller than half the distance between grid points, to make sure that we only used interpolation and not extrapolation. It could be that the implementation of the multireflection boundary technique caused extrapolation to happen on some points, due to the fact that this technique does not separate the distances between grid points and outer sphere points.

BIBLIOGRAPHY

[1] I. Ginzbourg and P. M. Adler, *Boundary flow condition analysis for three-dimensional lattice-boltzmann model,* Journal of Physics II France **4–2**, 191 (1994).

[2] R. S. Maier, R. S. Bernard, and D. Grunau, *Boundary conditions for the lattice-boltzmann method,* Physics of fluids **8–7**, 1788 (1996).

[3] D. P. Ziegler, *Boundary conditions for lattice-boltzmann simulations,* Journal of Statistical Physics **71**, 1171 (1993).

[4] M. Bouzidi, M. Firdaouss, and P. Lallemand, *Momentum transfer of a boltzmann-lattice fluid with boundaries,* Physics of fluids **13–11**, 3452 (2001).

[5] P. A. Skordos, *Initial and boundary conditions for the lattice-boltzmann method,* Physical Review E **48–6**, 4823 (1993).

[6] D. R. Noble, S. Chen, J. G. Georgiadis, and R. O. Buckius, *A consistent hydrodynamic boundary condition for the lattice boltzmann method,* Physics of fluids **7**, 203 (1995).

[7] O. Filippova and D. Hanel, *Grid refinement for lattice-bgk models,* Journal of Computational Physics **147–1**, 219 (1998).

[8] I. Ginzburg and D. d'Humières, *Multireflection boundary conditions for lattice boltzmann models,* Physical Review E **68–6**, 30 (2003).

[9] R. S. Maier and R. S. Bernard, *Lattice-boltzmann accuracy in pore-scale flow simulation,* Journal of Computational Physics **229**, 233 (2010).

[10] I. E. Agency, *World energy outlook 2013 factsheet,* , 2 (2013).

[11] N. Geographic, *Now-next: Meltdown-proof nukes,* (2011).

[12] G. McNamara and G. Zanetti, *Use of the boltzmann equation to simulate lattice gas automata,* Physical Review Letters **61**, 2332 (1988).

[13] J. Hardy, O. de Pazzis, and Y. Pomeau, *Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions,* Physical Review A **13**, 1949 (1976).

[14] A. A. Mohamad, *Lattice boltzmann method - fundamentals and engineering applications with computer codes,* Springer , 178 (2011).

[15] Y. B. Bao and J. Meskas, *Lattice boltzmann method for fluid simulations,* , 1 (2011).

[16] S. Chen and G. D. Doolen, *Lattice boltzmann method for fluid flows,* Annual Review of Fluid Mechanics **30**, 329 (1998).

[17] M. Rohde, *Cellular automata,* (2009).

[18] F. Higuera and J. Jiménez, *Boltzmann approach to lattice gas simulations,* Europhysics letters **9**, 663 (1989).

[19] P. Bhatnagar, E. P. Gross, and M. Krook, *A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems,* Physical Review **94**, 511 (1954).

[20] M. A. Gallivan, D. R. Noble, J. G. Georgiadis, and R. O. Buckius, *An evaluation of the bounce-back boundary condition for lattice boltzmann simulations,* International journal for numerical methods in fluids **25**, 249 (1997).

[21] *LBM workshop* (2011).

[22] B. Chun and A. J. C. Ladd, *Interpolated boundary condition for lattice boltzmann simulations of flows in narrow gaps,* Physical Review E **75** (2007).

[23] H. Hasimoto, *On the periodic fundamental solutions of the stokes equations and their application to viscous flow past a cubic array of spheres,* Journal of Fluid Mechanics **5**, 317 (1959).

[24] M. Rohde, J. J. Derksen, and H. E. A. van den Akker, *Volumetric method for calculating the flow around moving objects in lattice-boltzmann schemes,* Physical Review E **65**, 1 (2002).

[25] M. Rohde, *Extending the lattice-boltzmann method: Novel techniques for local grid refinement and boundary conditions,* , 190 (2004).

[26] B. Ahrenholz, J. Tolke, and M. Krafczyk, *Lattice-boltzmann simulations in reconstructed parametrized porous media,* International Journal of Computational Fluid Dynamics **20**, 369 (2006).

A

APPENDIX

A.1. D2Q9 HALF-WAY BOUNCE-BACK CODE

```
%D2Q9 model half-way bounce-back
%Grid size
xmax = 20;
ymax = xmax;
msize = xmax*ymax;

%Weighting factors
n1=4/9;
n2=1/9;
n3=1/36;
c_sqrd = 1/3;

%Benchmark variables
Re = 0.5;
vis = 1/6;
rho = 1;
tau = 1;

%Grid initial density D2Q9 (Maxwell-Boltzmann)
initial_density =   [1/9  1/36   1/9    1/36     1/9     1/36    1/9    1/36     4/9];

%Creating the solid obstacle
bol=zeros(xmax,ymax);
r=3;
nx = xmax/2; %center sphere x
ny = ymax/2; %center sphere y
for i=1:xmax
    for j=1:ymax
        if ((i-nx)^2+(j-ny)^2)<r^2
           bol(i,j)=((i-nx)^2+(j-ny)^2)<r^2;
        end
    end
end
D=2*r;

%Benchmark computations (we have not done these computations but they would need to be
%implemented here and should give us g, the drag coefficient.

%Reflecting properties for bounce-back
ci = [0:msize:msize*7];
ja = find(bol);

reflect = [ja+ci(1) ja+ci(2) ja+ci(3) ja+ci(4) ja+ci(5) ja+ci(6) ja+ci(7) ja+ci(8)];
reflected = [ja+ci(5) ja+ci(6) ja+ci(7) ja+ci(8)  ja+ci(1) ja+ci(2) ja+ci(3) ja+ci(4)];

%Define initial formula
```

```
density = ones(xmax,ymax);
n = ones(xmax,ymax);
for i = 1:length(initial_density)
    n(1:xmax,1:ymax,i) = initial_density(i).*density;
end
neq=n;

%Body force
FF1 = 3*[1 1 0 -1 -1 -1 0 1 0]*g;
FF2 = [n2 n3 n2 n3 n2 n3 n2 n3 n1];
F = FF1.*FF2;

%Initial while loop conditions
v_sqrd=0;
ts=0;
Reynold=1;
ReynoldA=1/2;

%Begin while loop
while ((((Reynold-ReynoldA)/ReynoldA>0.000001))|ts<5)

    %Calculating the density per node
    Density = sum(n,3);

    %While loop condition
    ReynoldA = mean(mean(v_x(nx+r+2,:)))*D/vis;
    ts = ts+1;

    %Calculating the velocity per node
    v_x =(sum(n(:,:,[1 2 8]),3)-sum(n(:,:,[4 5 6]),3))./Density;
    v_y =(sum(n(:,:,[2 3 4]),3)-sum(n(:,:,[6 7 8]),3))./Density;

    %Calculating the velocity vectors
    v_sqrd = v_x.^2+v_y.^2;
    vxplusvy = v_x+v_y;
    minvxplusvy = -v_x+v_y;
    minvxminvy = -vxplusvy;
    plusvxminvy = -minvxplusvy;

    %The equilibrium distribution function
    neq(:,:,9)=n1*Density.*(1                                    -v_sqrd/(2*c_sqrd));
    neq(:,:,1)=n2*Density.*(1+v_x/c_sqrd+0.5*(v_x/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,3)=n2*Density.*(1+v_y/c_sqrd+0.5*(v_y/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,5)=n2*Density.*(1-v_x/c_sqrd+0.5*(v_x/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,7)=n2*Density.*(1-v_y/c_sqrd+0.5*(v_y/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,2)=n3*Density.*(1+vxplusvy/c_sqrd+0.5*(vxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,4)=n3*Density.*(1+minvxplusvy/c_sqrd+0.5*(minvxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,6)=n3*Density.*(1+minvxminvy/c_sqrd+0.5*(minvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,8)=n3*Density.*(1+plusvxminvy/c_sqrd+0.5*(plusvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));

    %The normal collision setp (no bounce-back)
    n = n-(n-neq)/tau;

    %Applying the body force
    for i=1:9
        n(:,:,i) = n(:,:,i) + F(i);
    end

    %Initiating bounce-back
    BB = n(reflect);

    %The propagation step
    n(:,:,4)=n([2:xmax 1],[ymax 1:ymax-1],4);
    n(:,:,3)=n(:,[ymax 1:ymax-1],3);
    n(:,:,2)=n([xmax 1:xmax-1],[ymax 1:ymax-1],2);
    n(:,:,5)=n([2:xmax 1],:,5);
    n(:,:,1)=n([xmax 1:xmax-1],:,1);
    n(:,:,6)=n([2:xmax 1],[2:ymax 1],6);
    n(:,:,7)=n(:,[2:ymax 1],7);
    n(:,:,8)=n([xmax 1:xmax-1],[2:ymax 1],8);
```

```
    %Bounceback
    n(reflected) = BB;

    %Calculating the Reynolds number
    Reynold = mean(mean(v_x(nx+r+2,:)))*D/vis;
end
```

A.2. D3Q19 HALF-WAY BOUNCE-BACK CODE

```
%D3Q19 model half-way bounce-back
%Grid size
xmax = 128;
ymax = xmax;
zmax = xmax;
msize = xmax*ymax*zmax;

%Weighting factors
n1 = 1/3;
n2 = 1/18;
n3 = 1/36;
c_sqrd = 1/3;

%Benchmark variables
Re = 0.5;
vis = 1/6;
rho =1;
tau = 1;

%Grid initial density D3Q19
initial_density = [1/3 1/18 1/18 1/18 1/18 1/18 1/18 1/36 1/36 1/36 1/36 1/36 1/36 1/36 1/36
1/36 1/36 1/36 1/36];

%Creating the solid obstacle
bol =zeros(xmax,ymax,zmax);
r=6;
nx =xmax/2;
ny = ymax/2;
nz = zmax/2;
for i = 1:xmax
    for j = 1:ymax
        for k = 1:zmax
           bol(i,j,k)=((i-nx)^2+(j-ny)^2+(k-nz)^2)<r^2;
        end
    end
end
D=2*r;

%Benchmark computations
V_f = msize-1/6*pi*D^3;
phi = pi*D^3/(6*xmax^3);
g = 3*pi*vis^2*Re/(V_f*(1-1.7601*phi^(1/3)+phi-1.5593*phi^2));

%Reflecting properties for bounce-back
ci = [0:msize:msize*19];
ja = find(bol);

reflect = [ja+ci(2) ja+ci(3) ja+ci(4) ja+ci(5) ja+ci(6) ja+ci(7) ja+ci(8) ja+ci(9) ja+ci(10)
ja+ci(11) ja+ci(12) ja+ci(13)  ja+ci(14) ja+ci(15) ja+ci(16) ja+ci(17) ja+ci(18) ja+ci(19)];
reflected = [ja+ci(3) ja+ci(2) ja+ci(5) ja+ci(4) ja+ci(7) ja+ci(6) ja+ci(11) ja+ci(10) ja+ci(9)
ja+ci(8) ja+ci(15) ja+ci(14)  ja+ci(13) ja+ci(12) ja+ci(19) ja+ci(18) ja+ci(17) ja+ci(16)];

%Define initial formula
density=ones(xmax,ymax,zmax);
n=ones(xmax,ymax,zmax);
for i = 1:length(initial_density)
    n(1:xmax,1:ymax,1:zmax,i) = initial_density(i).*density;
end
neq=n;
```

```
%Body force
FF1 = 3*[0 0 0 0 0 1 -1 1 1 -1 -1 1 1 -1 -1 0 0 0 0]*g;
FF2 = [n1 n2*ones(1,6) n3*ones(1,12)];
F = FF1.*FF2;

%Initial while loop conditions
v_x=0;
ts=0;
Reynold=1;
ReynoldA=1/2;

%Begin while loop
while ((((Reynold-ReynoldA)/ReynoldA>0.000001))|ts<5)

    %Calculating the density per node
    Density = sum(n,4);

    %While loop condition
    ReynoldA = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
    ts = ts+1;

    %Calculating the velocity per node
    v_x =(sum(n(:,:,:,[6 8 9 13 12]),4)-sum(n(:,:,:,[7 11 10 14 15]),4))./Density;
    v_y =(sum(n(:,:,:,[4 8 10 16 17]),4)-sum(n(:,:,:,[5 9 11 18 19]),4))./Density;
    v_z =(sum(n(:,:,:,[2 12 14 16 18]),4)-sum(n(:,:,:,[3 13 15 17 19]),4))./Density;

    %Calculating the velocity vectors
    v_sqrd = v_x.^2+v_y.^2+v_z.^2;
    vxplusvy = v_x+v_y;
    minvxplusvy = -v_x+v_y;
    minvxminvy = -vxplusvy;
    plusvxminvy = -minvxplusvy;
    vxplusvz = v_x+v_z;
    vxminvz = v_x-v_z;
    minvxminvz = -vxplusvz;
    vzminvx = -vxminvz;
    vyplusvz = v_y+v_z;
    vyminvz = v_y-v_z;
    minvyminvz = -vyplusvz;
    vzminvy = -vyminvz;

    %The equilibrium distribuiton function
    neq(:,:,:,1)=n1*Density.*(1-                                        v_sqrd/(2*c_sqrd));
    neq(:,:,:,2)=n2*Density.*(1 + v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,3)=n2*Density.*(1 - v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,4)=n2*Density.*(1 + v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,5)=n2*Density.*(1 - v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,6)=n2*Density.*(1 + v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,7)=n2*Density.*(1 - v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,8)=n3*Density.*(1 + vxplusvy/c_sqrd+0.5*(vxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,9)=n3*Density.*(1 + plusvxminvy/c_sqrd+0.5*(plusvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,10)=n3*Density.*(1 + minvxplusvy/c_sqrd+0.5*(minvxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,11)=n3*Density.*(1 + minvxminvy/c_sqrd+0.5*(minvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,12)=n3*Density.*(1 + vxplusvz/c_sqrd+0.5*(vxplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,13)=n3*Density.*(1 + vxminvz/c_sqrd+0.5*(vxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,14)=n3*Density.*(1 + vzminvx/c_sqrd+0.5*(vzminvx/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,15)=n3*Density.*(1 + minvxminvz/c_sqrd+0.5*(minvxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,16)=n3*Density.*(1 + vyplusvz/c_sqrd+0.5*(vyplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,17)=n3*Density.*(1 + vyminvz/c_sqrd+0.5*(vyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,18)=n3*Density.*(1 + vzminvy/c_sqrd+0.5*(vzminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,19)=n3*Density.*(1 + minvyminvz/c_sqrd+0.5*(minvyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));

    %The normal collision step (no bounce-back)
    n = n-(n-neq)/tau;

    %Applying the body force
    for i=1:19
        n(:,:,:,i) = n(:,:,:,i) + F(i);
    end

    %Initiating bounce-back
```

```
    BB=n(reflect);

    %The propagation step
    n(:,:,:,2)=n(:,:,[zmax 1:zmax-1],2);
    n(:,:,:,3)=n(:,:,[2:zmax 1],3);
    n(:,:,:,4)=n(:,[ymax 1:ymax-1],:,4);
    n(:,:,:,5)=n(:,[2:ymax 1],:,5);
    n(:,:,:,6)=n([xmax 1:xmax-1],:,:,6);
    n(:,:,:,7)=n([2:xmax 1],:,:,7);
    n(:,:,:,8)= n([xmax 1:xmax-1],[ymax 1:ymax-1],:,8);
    n(:,:,:,9)= n([xmax 1:xmax-1],[2:ymax 1],:,9);
    n(:,:,:,10)=n([2:xmax 1],[ymax 1:ymax-1],:,10);
    n(:,:,:,11)=n([2:xmax 1],[2:ymax 1],:,11);
    n(:,:,:,12)=n([xmax 1:xmax-1],:,[zmax 1:zmax-1],12);
    n(:,:,:,13)=n([xmax 1:xmax-1],:,[2:zmax 1],13);
    n(:,:,:,14)=n([2:xmax 1],:,[zmax 1:zmax-1],14);
    n(:,:,:,15)=n([2:xmax 1],:,[2:zmax 1],15);
    n(:,:,:,16)=n(:,[ymax 1:ymax-1],[zmax 1:zmax-1],16);
    n(:,:,:,17)=n(:,[ymax 1:ymax-1],[2:zmax 1],17);
    n(:,:,:,18)=n(:,[2:ymax 1],[zmax 1:zmax-1],18);
    n(:,:,:,19)=n(:,[2:ymax 1],[2:zmax 1],19);

    %Bounceback
    n(reflected) = BB;

    %Calculating the Reynolds number
    Reynold = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
end
```

A.3. MATLAB FORMULA FOR FINDING ALL RELEVANT LINE-SPHERE INTERSECTIONS

```
function [d, plaatsx, plaatsy, plaatsz, plaatsxtoe, plaatsytoe, plaatsztoe, plaatsxvorig,
plaatsyvorig, plaatszvorig, richting, richtingsterr] =
distancedried(Ix, Iy, Iz, Ox, Oy, Oz, nx, ny, nz, r, richtingn, richtingster)
dx3=0;
plaatsx1 = 0;
plaatsy1 = 0;
plaatsz1 = 0;
snelheidx1 = 0;
snelheidy1 = 0;
snelheidz1 = 0;
richting1 = 0;
 for i=1:length(Ix)
     for j = 1:length(Ox)
         for k = 1:length(Oy)
             for l = 1:length(Oz)
                root = (Ix(i)*(Ox(j)-nx)+Iy(i)*(Oy(k)-ny)+Iz(i)*(Oz(l)-nz))^2-(Ox(j)-nx)^2-
                (Oy(k)-ny)^2-(Oz(l)-nz)^2+r^2;
                    if root>=0
                        dxplus = -(Ix(i)*(Ox(j)-nx)+Iy(i)*(Oy(k)-ny)+Iz(i)*(Oz(l)-nz))+sqrt(root);
                        dxmin = -(Ix(i)*(Ox(j)-nx)+Iy(i)*(Oy(k)-ny)+Iz(i)*(Oz(l)-nz))-sqrt(root);
                        if((dxplus>=0)&&(dxplus<=1))
                            dx3 = [dx3 dxplus];
                            plaatsx1 = [plaatsx1 Ox(j)];
                            plaatsy1 = [plaatsy1 Oy(k)];
                            plaatsz1 = [plaatsz1 Oz(l)];
                            snelheidx1 = [snelheidx1 Ix(i)];
                            snelheidy1 = [snelheidy1 Iy(i)];
                            snelheidz1 = [snelheidz1 Iz(i)];
                            richting1 = [richting1 i];
                        elseif((dxmin>=0)&&(dxmin<=1))
                            dx3 = [dx3 dxmin];
                            plaatsx1 = [plaatsx1 Ox(j)];
                            plaatsy1 = [plaatsy1 Oy(k)];
                            plaatsz1 = [plaatsz1 Oz(l)];
                            snelheidx1 = [snelheidx1 Ix(i)];
                            snelheidy1 = [snelheidy1 Iy(i)];
```

```
                        snelheidz1 = [snelheidz1 Iz(i)];
                        richting1 = [richting1 i];
                    end
                end
            end
        end
    end
end
plaatsx20 = plaatsx1(2:length(plaatsx1));
plaatsy20 = plaatsy1(2:length(plaatsy1));
plaatsz20 = plaatsz1(2:length(plaatsz1));
snelheidx20 = snelheidx1(2:length(snelheidx1));
snelheidy20 = snelheidy1(2:length(snelheidy1));
snelheidz20 = snelheidz1(2:length(snelheidz1));
d20 = dx3(2:length(dx3));
richting20 = richting1(2:length(richting1));

Z = sum(richting20==1);
plaatsx = plaatsx20((Z+1):length(plaatsx20));
plaatsy = plaatsy20((Z+1):length(plaatsy20));
plaatsz = plaatsz20((Z+1):length(plaatsz20));
snelheidx = snelheidx20((Z+1):length(snelheidx20));
snelheidy = snelheidy20((Z+1):length(snelheidy20));
snelheidz = snelheidz20((Z+1):length(snelheidz20));
d = d20((Z+1):length(d20));
richting = richting20((Z+1):length(richting20));

plaatsx1 = 0;
plaatsx2 = 0;
plaatsy1 = 0;
plaatsy2 = 0;
plaatsz1 = 0;
plaatsz2 = 0;
for i = 1:length(plaatsx)
   if snelheidx(i)>0
       plaatsx1 = [plaatsx1 plaatsx(i)-2];
       plaatsx2 = [plaatsx2 plaatsx(i)-1];
   elseif snelheidx(i)<0
       plaatsx1 = [plaatsx1 plaatsx(i)+2];
       plaatsx2 = [plaatsx2 plaatsx(i)+1];
   else
       plaatsx1 = [plaatsx1 plaatsx(i)];
       plaatsx2 = [plaatsx2 plaatsx(i)];
   end
    if snelheidy(i)>0
       plaatsy1 = [plaatsy1 plaatsy(i)-2];
       plaatsy2 = [plaatsy2 plaatsy(i)-1];
   elseif snelheidy(i)<0
       plaatsy1 = [plaatsy1 plaatsy(i)+2];
       plaatsy2 = [plaatsy2 plaatsy(i)+1];
   else
       plaatsy1 = [plaatsy1 plaatsy(i)];
       plaatsy2 = [plaatsy2 plaatsy(i)];
    end
    if snelheidz(i)>0
       plaatsz1 = [plaatsz1 plaatsz(i)-2];
       plaatsz2 = [plaatsz2 plaatsz(i)-1];
   elseif snelheidz(i)<0
       plaatsz1 = [plaatsz1 plaatsz(i)+2];
       plaatsz2 = [plaatsz2 plaatsz(i)+1];
   else
       plaatsz1 = [plaatsz1 plaatsz(i)];
       plaatsz2 = [plaatsz2 plaatsz(i)];
    end
end
plaatsxtoe = plaatsx1(2:length(plaatsx1));
plaatsytoe = plaatsy1(2:length(plaatsy1));
plaatsztoe = plaatsz1(2:length(plaatsz1));
plaatsxvorig = plaatsx2(2:length(plaatsx2));
plaatsyvorig = plaatsy2(2:length(plaatsy2));
plaatszvorig = plaatsz2(2:length(plaatsz2));
```

```
 richtster = 0;
 for i = 1:length(richting)
    for j = 1:length(richtingn)
        if richting(i)==richtingn(j)
             richtster = [richtster richtingster(j)];
        end
    end
 end
 richtingsterr = richtster(2:length(richtster));
 return
end
```

A.4. D3Q19 LINEAR INTERPOLATION CODE

```
%D3Q19 model linear interpolation
%Grid size
xmax = 12;
ymax = xmax;
zmax = xmax;
msize = xmax*ymax*zmax;
Ox = [1:xmax];
Oy = Ox;
Oz = Oy;

%Weighting factors
n1 = 1/3;
n2 = 1/18;
n3 = 1/36;
c_sqrd = 1/3;

%Benchmark variables
Re = 0.5;
vis=1/6;
rho =1;
tau=1;

%Grid initial density and velocities D3Q19
initial_density = [1/3 1/18 1/18 1/18 1/18 1/18 1/18 1/36 1/36 1/36 1/36 1/36 1/36 1/36 1/36
1/36 1/36 1/36 1/36];
Ix = [0 0  0 0 0  1 -1 0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) 0.5*sqrt(2)
-0.5*sqrt(2) -0.5*sqrt(2) 0 0 0 0 ];
Iy = [0 0  0 1 -1 0  0 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0           0
0 0 0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2) ];
Iz = [0 1 -1 0 0  0  0 0           0            0           0            0.5*sqrt(2) -0.5*sqrt(2)
0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) ];
richtingn =    [2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19];
richtingster = [3 2 5 4 7 6 11 10 9 8 15 14 13 12 19 18 17 16];

%Benchmark computations for a sphere with radius r, and center at position (nx, ny, nz)
r=3;
nx =xmax/2;
ny = ymax/2;
nz = zmax/2;
D=2*r;
V_f = msize-1/6*pi*D^3;
phi = pi*D^3/(6*xmax^3);
g = 3*pi*vis^2*Re/(V_f*(1-1.7601*phi^(1/3)+phi-1.5593*phi^2));

%Formula for calculating the location of the boundary as seen from the nodes.
[q, plaatsx, plaatsy, plaatsz, plaatsxtoe, plaatsytoe, plaatsztoe, plaatsxvorig, plaatsyvorig, plaatszvorig,
richting, richtingsterr] = distancedried(Ix, Iy, Iz, Ox, Oy, Oz, nx, ny, nz, r, richtingn, richtingster);

%Define initial formula
density=ones(xmax,ymax,zmax);
n=ones(xmax,ymax,zmax);
for i = 1:length(initial_density)
    n(1:xmax,1:ymax,1:zmax,i) = initial_density(i).*density;
end
```

```
neq=n;

%Body force
FF1 = 3*[0 0 0 0 0 1 -1 1 1 -1 -1 1 1 -1 -1 0 0 0 0]*g;
FF2 = [n1 n2*ones(1,6) n3*ones(1,12)];
F = FF1.*FF2;

%Initial while loop conditions
v_x=0;
ts=0;
Reynold=1;
ReynoldA=1/2;

%Begin while loop
while ((((Reynold-ReynoldA)/ReynoldA>0.000001))|ts<5)

    %Calculating the density per node
    Density = sum(n,4);

    %While loop condition
    ReynoldA = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
    ts = ts+1;

    %Calculating velocity per node
    v_x =(sum(n(:,:,:,[6 8 9 13 12]),4)-sum(n(:,:,:,[7 11 10 14 15]),4))./Density;
    v_y =(sum(n(:,:,:,[4 8 10 16 17]),4)-sum(n(:,:,:,[5 9 11 18 19]),4))./Density;
    v_z =(sum(n(:,:,:,[2 12 14 16 18]),4)-sum(n(:,:,:,[3 13 15 17 19]),4))./Density;

    %Calculating the velocity vectors
    v_sqrd = v_x.^2+v_y.^2+v_z.^2;
    vxplusvy = v_x+v_y;
    minvxplusvy = -v_x+v_y;
    minvxminvy = -vxplusvy;
    plusvxminvy = -minvxplusvy;
    vxplusvz = v_x+v_z;
    vxminvz = v_x-v_z;
    minvxminvz = -vxplusvz;
    vzminvx = -vxminvz;
    vyplusvz = v_y+v_z;
    vyminvz = v_y-v_z;
    minvyminvz = -vyplusvz;
    vzminvy = -vyminvz;

    %The equilibrium distribution function
    neq(:,:,:,1)=n1*Density.*(1-                                          v_sqrd/(2*c_sqrd));
    neq(:,:,:,2)=n2*Density.*(1 + v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,3)=n2*Density.*(1 - v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,4)=n2*Density.*(1 + v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,5)=n2*Density.*(1 - v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,6)=n2*Density.*(1 + v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,7)=n2*Density.*(1 - v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,8) =n3*Density.*(1 + vxplusvy/c_sqrd+0.5*(vxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,9) =n3*Density.*(1 + plusvxminvy/c_sqrd+0.5*(plusvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,10)=n3*Density.*(1 + minvxplusvy/c_sqrd+0.5*(minvxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,11)=n3*Density.*(1 + minvxminvy/c_sqrd+0.5*(minvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,12)=n3*Density.*(1 + vxplusvz/c_sqrd+0.5*(vxplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,13)=n3*Density.*(1 + vxminvz/c_sqrd+0.5*(vxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,14)=n3*Density.*(1 + vzminvx/c_sqrd+0.5*(vzminvx/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,15)=n3*Density.*(1 + minvxminvz/c_sqrd+0.5*(minvxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,16)=n3*Density.*(1 + vyplusvz/c_sqrd+0.5*(vyplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,17)=n3*Density.*(1 + vyminvz/c_sqrd+0.5*(vyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,18)=n3*Density.*(1 + vzminvy/c_sqrd+0.5*(vzminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,19)=n3*Density.*(1 + minvyminvz/c_sqrd+0.5*(minvyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));

    %The normal collision step (no bounce-back)
    n = n-(n-neq)/tau;

    %Applying the body force
    for i=1:19
        n(:,:,:,i) = n(:,:,:,i) + F(i);
    end
```

```
    %Initiating the boundary condition
    BB = zeros(1,length(q));
    for i = 1:length(q)
        if q(i)<0.5
            BB(i) = 2*q(i)*n(plaatsx(i), plaatsy(i), plaatsz(i), richting(i))+
            (1-2*q(i))*n(plaatsxvorig(i), plaatsyvorig(i), plaatszvorig(i), richting(i));
        elseif q(i)>=0.5
            BB(i) = 1/(2*q(i))*n(plaatsx(i), plaatsy(i), plaatsz(i), richting(i))+
            (2*q(i)-1)/(2*q(i))*n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i));
        end
    end

    %The propagation step
    n(:,:,:,2)=n(:,:,[zmax 1:zmax-1],2);
    n(:,:,:,3)=n(:,:,[2:zmax 1],3);
    n(:,:,:,4)=n(:,[ymax 1:ymax-1],:,4);
    n(:,:,:,5)=n(:,[2:ymax 1],:,5);
    n(:,:,:,6)=n([xmax 1:xmax-1],:,:,6);
    n(:,:,:,7)=n([2:xmax 1],:,:,7);
    n(:,:,:,8)= n([xmax 1:xmax-1],[ymax 1:ymax-1],:,8);
    n(:,:,:,9)= n([xmax 1:xmax-1],[2:ymax 1],:,9);
    n(:,:,:,10)=n([2:xmax 1],[ymax 1:ymax-1],:,10);
    n(:,:,:,11)=n([2:xmax 1],[2:ymax 1],:,11);
    n(:,:,:,12)=n([xmax 1:xmax-1],:,[zmax 1:zmax-1],12);
    n(:,:,:,13)=n([xmax 1:xmax-1],:,[2:zmax 1],13);
    n(:,:,:,14)=n([2:xmax 1],:,[zmax 1:zmax-1],14);
    n(:,:,:,15)=n([2:xmax 1],:,[2:zmax 1],15);
    n(:,:,:,16)=n(:,[ymax 1:ymax-1],[zmax 1:zmax-1],16);
    n(:,:,:,17)=n(:,[ymax 1:ymax-1],[2:zmax 1],17);
    n(:,:,:,18)=n(:,[2:ymax 1],[zmax 1:zmax-1],18);
    n(:,:,:,19)=n(:,[2:ymax 1],[2:zmax 1],19);

    %Applying the linear interpolation boundary condition
    for i=1:length(q)
        n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i)) = BB(i);
    end

    %Calculating the Reynolds number
    Reynold = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
end
```

A.5. D3Q19 QUADRATIC INTERPOLATION CODE

```
%D3Q19 model quadratic interpolation
%Grid size
xmax = 12;
ymax = xmax;
zmax = xmax;
msize = xmax*ymax*zmax;
Ox = [1:xmax];
Oy = Ox;
Oz = Oy;

%Weighting factors
n1 = 1/3;
n2 = 1/18;
n3 = 1/36;
c_sqrd = 1/3;

%Benchmark variables
Re = 0.5;
tau=1;
vis=1/6;
rho =1;

%Grid initial density and velocities D3Q19
initial_density = [1/3 1/18 1/18 1/18 1/18 1/18 1/18 1/36 1/36 1/36 1/36 1/36 1/36 1/36 1/36
1/36 1/36 1/36 1/36];
```

```
Ix = [0 0 0 0 0 1 -1 0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) 0.5*sqrt(2)
-0.5*sqrt(2) -0.5*sqrt(2) 0 0 0 0];
Iy = [0 0 0 1 -1 0 0 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0 0 0 0 0.5*sqrt(2)
0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2)];
Iz = [0 1 -1 0 0 0 0 0 0 0 0 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2)
-0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2)];
richtingn =    [2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19];
richtingster = [3 2 5 4 7 6 11 10 9 8 15 14 13 12 19 18 17 16];

%Benchmark computations for a sphere with radius r, and center at position (nx, ny, nz)
r=3;
nx = xmax/2;
ny = ymax/2;
nz = zmax/2;
D=2*r;
V_f = msize-1/6*pi*D^3;
phi = pi*D^3/(6*xmax^3);
g = 3*pi*vis^2*Re/(V_f*(1-1.7601*phi^(1/3)+phi-1.5593*phi^2));

%Formula for calculating the location of the boundary as seen from the nodes
[q, plaatsx, plaatsy, plaatsz, plaatsxtoe, plaatsytoe, plaatsztoe, plaatsxvorig, plaatsyvorig, plaatszvori
richting, richtingsterr] = distancedried(Ix, Iy, Iz, Ox, Oy, Oz, nx, ny, nz, r, richtingn, richtingster);

%Define initial formula
density=ones(xmax,ymax,zmax);
n=ones(xmax,ymax,zmax);
for i = 1:length(initial_density)
    n(1:xmax,1:ymax,1:zmax,i) = initial_density(i).*density;
end
neq=n;

%Body force
FF1 = 3*[0 0 0 0 0 1 -1 1 1 -1 -1 1 1 -1 -1 0 0 0 0]*g;
FF2 = [n1 n2*ones(1,6) n3*ones(1,12)];
F = FF1.*FF2;

%Initial while loop conditions
v_x = 0;
ts=0;
Reynold=1;
ReynoldA=1/2;

% Begin while loop
while ((((Reynold-ReynoldA)/ReynoldA>0.000001))|ts<5)

    %Calculating the density per node
    Density = sum(n,4);

    %While loop condition
    ReynoldA = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
    ts = ts+1;

    %Calculating the velocity per node
    v_x =(sum(n(:,:,:,[6 8 9 13 12]),4)-sum(n(:,:,:,[7 11 10 14 15]),4))./Density;
    v_y =(sum(n(:,:,:,[4 8 10 16 17]),4)-sum(n(:,:,:,[5 9 11 18 19]),4))./Density;
    v_z =(sum(n(:,:,:,[2 12 14 16 18]),4)-sum(n(:,:,:,[3 13 15 17 19]),4))./Density;

    %Calculating the velocity vectors
    v_sqrd = v_x.^2+v_y.^2+v_z.^2;
    vxplusvy = v_x+v_y;
    minvxplusvy = -v_x+v_y;
    minvxminvy = -vxplusvy;
    plusvxminvy = -minvxplusvy;
    vxplusvz = v_x+v_z;
    vxminvz = v_x-v_z;
    minvxminvz = -vxplusvz;
    vzminvx = -vxminvz;
    vyplusvz = v_y+v_z;
    vyminvz = v_y-v_z;
    minvyminvz = -vyplusvz;
    vzminvy = -vyminvz;
```

```
%The equilibrium distribution function
neq(:,:,:,1)=n1*Density.*(1-                                          v_sqrd/(2*c_sqrd));
neq(:,:,:,2)=n2*Density.*(1 + v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,3)=n2*Density.*(1 - v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,4)=n2*Density.*(1 + v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,5)=n2*Density.*(1 - v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,6)=n2*Density.*(1 + v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,7)=n2*Density.*(1 - v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -     v_sqrd/(2*c_sqrd));
neq(:,:,:,8) =n3*Density.*(1 + vxplusvy/c_sqrd+0.5*(vxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,9) =n3*Density.*(1 + plusvxminvy/c_sqrd+0.5*(plusvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,10)=n3*Density.*(1 + minvxplusvy/c_sqrd+0.5*(minvxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,11)=n3*Density.*(1 + minvxminvy/c_sqrd+0.5*(minvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,12)=n3*Density.*(1 + vxplusvz/c_sqrd+0.5*(vxplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,13)=n3*Density.*(1 + vxminvz/c_sqrd+0.5*(vxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,14)=n3*Density.*(1 + vzminvx/c_sqrd+0.5*(vzminvx/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,15)=n3*Density.*(1 + minvxminvz/c_sqrd+0.5*(minvxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,16)=n3*Density.*(1 + vyplusvz/c_sqrd+0.5*(vyplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,17)=n3*Density.*(1 + vyminvz/c_sqrd+0.5*(vyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,18)=n3*Density.*(1 + vzminvy/c_sqrd+0.5*(vzminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
neq(:,:,:,19)=n3*Density.*(1 + minvyminvz/c_sqrd+0.5*(minvyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));

%The normal collision step (no bounce-back)
n = n-(n-neq)/tau;

%Applying the body force
for i=1:19
    n(:,:,:,i) = n(:,:,:,i) + F(i);
end

%Initiating the boundary condition
BB = zeros(1,length(q));
for i = 1:length(q)
    if q(i)<0.5
        BB(i) = q(i)*(2*q(i)+1)*n(plaatsx(i), plaatsy(i), plaatsz(i), richting(i))+
        (1-4*(q(i))^2)*n(plaatsxvorig(i), plaatsyvorig(i), plaatszvorig(i), richting(i))-
        q(i)*(1-2*q(i))*n(plaatsxtoe(i), plaatsytoe(i), plaatsztoe(i), richting(i));
    elseif q(i)>=0.5
        BB(i) = 1/(q(i)*(2*q(i)+1))*n(plaatsx(i), plaatsy(i), plaatsz(i), richting(i))+
        (2*q(i)-1)/q(i)*n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i))+
        (1-2*q(i))/(1+2*q(i))*n(plaatsxvorig(i), plaatsyvorig(i), plaatszvorig(i), richtingsterr(i));
    end
end

%The propagation step
n(:,:,:,2)=n(:,:,[zmax 1:zmax-1],2);
n(:,:,:,3)=n(:,:,[2:zmax 1],3);
n(:,:,:,4)=n(:,[ymax 1:ymax-1],:,4);
n(:,:,:,5)=n(:,[2:ymax 1],:,5);
n(:,:,:,6)=n([xmax 1:xmax-1],:,:,6);
n(:,:,:,7)=n([2:xmax 1],:,:,7);
n(:,:,:,8)= n([xmax 1:xmax-1],[ymax 1:ymax-1],:,8);
n(:,:,:,9)= n([xmax 1:xmax-1],[2:ymax 1],:,9);
n(:,:,:,10)=n([2:xmax 1],[ymax 1:ymax-1],:,10);
n(:,:,:,11)=n([2:xmax 1],[2:ymax 1],:,11);
n(:,:,:,12)=n([xmax 1:xmax-1],:,[zmax 1:zmax-1],12);
n(:,:,:,13)=n([xmax 1:xmax-1],:,[2:zmax 1],13);
n(:,:,:,14)=n([2:xmax 1],:,[zmax 1:zmax-1],14);
n(:,:,:,15)=n([2:xmax 1],:,[2:zmax 1],15);
n(:,:,:,16)=n(:,[ymax 1:ymax-1],[zmax 1:zmax-1],16);
n(:,:,:,17)=n(:,[ymax 1:ymax-1],[2:zmax 1],17);
n(:,:,:,18)=n(:,[2:ymax 1],[zmax 1:zmax-1],18);
n(:,:,:,19)=n(:,[2:ymax 1],[2:zmax 1],19);

%Applying the linear interpolation boundary condition
for i=1:length(q)
    n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i)) = BB(i);
end

%Calculating the Reynolds number
Reynold = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
```

```
end
```

A.6. D3Q19 MULTIREFLECTION CODE

```
%D3Q19 model multireflection boundary conditions
%Grid size
xmax = 128;
ymax = xmax;
zmax = xmax;
msize = xmax*ymax*zmax;
Ox = [1:xmax];
Oy = Ox;
Oz = Oy;

%Weighting factors
n1 = 1/3;
n2 = 1/18;
n3 = 1/36;
c_sqrd = 1/3;

%Benchmark variables
Re = 0.5;
vis=1/6;
rho =1;
tau=1;

%Grid initial density and velocities D3Q19
initial_density = [1/3 1/18 1/18 1/18 1/18 1/18 1/18 1/36 1/36 1/36 1/36 1/36 1/36 1/36 1/36
1/36 1/36 1/36 1/36];
Ix = [0 0  0 0 0  1 -1 0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) 0.5*sqrt(2)
-0.5*sqrt(2) -0.5*sqrt(2) 0 0 0 0 ];
Iy = [0 0  0 1 -1 0  0 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0           0
0 0 0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) -0.5*sqrt(2) ];
Iz = [0 1 -1 0 0  0  0 0           0           0           0           0.5*sqrt(2) -0.5*sqrt(2)
0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) 0.5*sqrt(2) -0.5*sqrt(2) ];
richtingn =    [2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19];
richtingster = [3 2 5 4 7 6 11 10 9 8 15 14 13 12 19 18 17 16];

%Benchmark computations for a sphere with radius r, and center at position (nx, ny, nz)
r=6;
nx =xmax/2;
ny = ymax/2;
nz = zmax/2;
D=2*r;
V_f = msize-1/6*pi*D^3;
phi = pi*D^3/(6*xmax^3);
g = 3*pi*vis^2*Re/(V_f*(1-1.7601*phi^(1/3)+phi-1.5593*phi^2));

%Formula for calculating the location of the boundary as seen from the nodes
[q, plaatsx, plaatsy, plaatsz, plaatsxtoe, plaatsytoe, plaatsztoe, plaatsxvorig, plaatsyvorig, plaatszvori
richting, richtingsterr] = distancedried(Ix, Iy, Iz, Ox, Oy, Oz, nx, ny, nz, r, richtingn, richtingster);

%Define initial formula
density=ones(xmax,ymax,zmax);
n=ones(xmax,ymax,zmax);
for i = 1:length(initial_density)
    n(1:xmax,1:ymax,1:zmax,i) = initial_density(i).*density;
end
neq=n;

%Body force
FF1 = 3*[0 0 0 0 0 1 -1 1 1 -1 -1 1 1 -1 -1 0 0 0 0]*g;
FF2 = [n1 n2*ones(1,6) n3*ones(1,12)];
F = FF1.*FF2;

%Initial while loop conditions
ts=0;
v_x=0;
Reynold=1;
```

```
ReynoldA=1/2;

%Begin while loop
while ((((Reynold-ReynoldA)/ReynoldA>0.000001))|ts<5)

    %Calculating the density per node
    Density = sum(n,4);

    %While loop condition
    ReynoldA = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
    ts = ts+1;

    %Calculating the velocity per node
    v_x =(sum(n(:,:,:,[6 8 9 13 12]),4)-sum(n(:,:,:,[7 11 10 14 15]),4))./Density;
    v_y =(sum(n(:,:,:,[4 8 10 16 17]),4)-sum(n(:,:,:,[5 9 11 18 19]),4))./Density;
    v_z =(sum(n(:,:,:,[2 12 14 16 18]),4)-sum(n(:,:,:,[3 13 15 17 19]),4))./Density;

    %Calculating the velocity vectors
    v_sqrd = v_x.^2+v_y.^2+v_z.^2;
    vxplusvy = v_x+v_y;
    minvxplusvy = -v_x+v_y;
    minvxminvy = -vxplusvy;
    plusvxminvy = -minvxplusvy;
    vxplusvz = v_x+v_z;
    vxminvz = v_x-v_z;
    minvxminvz = -vxplusvz;
    vzminvx = -vxminvz;
    vyplusvz = v_y+v_z;
    vyminvz = v_y-v_z;
    minvyminvz = -vyplusvz;
    vzminvy = -vyminvz;

    %The equilibrium distribution function
    neq(:,:,:,1)=n1*Density.*(1-                                        v_sqrd/(2*c_sqrd));
    neq(:,:,:,2)=n2*Density.*(1 + v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,3)=n2*Density.*(1 - v_z/c_sqrd + 0.5*(v_z/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,4)=n2*Density.*(1 + v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,5)=n2*Density.*(1 - v_y/c_sqrd + 0.5*(v_y/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,6)=n2*Density.*(1 + v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,7)=n2*Density.*(1 - v_x/c_sqrd + 0.5*(v_x/c_sqrd).^2 -    v_sqrd/(2*c_sqrd));
    neq(:,:,:,8) =n3*Density.*(1 + vxplusvy/c_sqrd+0.5*(vxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,9) =n3*Density.*(1 + plusvxminvy/c_sqrd+0.5*(plusvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,10)=n3*Density.*(1 + minvxplusvy/c_sqrd+0.5*(minvxplusvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,11)=n3*Density.*(1 + minvxminvy/c_sqrd+0.5*(minvxminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,12)=n3*Density.*(1 + vxplusvz/c_sqrd+0.5*(vxplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,13)=n3*Density.*(1 + vxminvz/c_sqrd+0.5*(vxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,14)=n3*Density.*(1 + vzminvx/c_sqrd+0.5*(vzminvx/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,15)=n3*Density.*(1 + minvxminvz/c_sqrd+0.5*(minvxminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,16)=n3*Density.*(1 + vyplusvz/c_sqrd+0.5*(vyplusvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,17)=n3*Density.*(1 + vyminvz/c_sqrd+0.5*(vyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,18)=n3*Density.*(1 + vzminvy/c_sqrd+0.5*(vzminvy/c_sqrd).^2-v_sqrd/(2*c_sqrd));
    neq(:,:,:,19)=n3*Density.*(1 + minvyminvz/c_sqrd+0.5*(minvyminvz/c_sqrd).^2-v_sqrd/(2*c_sqrd));

    %The normal collision step (no bounce-back)
    n = n-(n-neq)/tau;

    %Initiating the boundary condition
    BB = zeros(1,length(q));
    for i = 1:length(q)
        BB(i) = n(plaatsx(i), plaatsy(i), plaatsz(i), richting(i))+
        (1-2*q(i)-2*(q(i))^2)/(1+q(i))^2*n(plaatsxvorig(i), plaatsyvorig(i), plaatszvorig(i), richting(i))+
        (q(i))^2/(1+q(i))^2*n(plaatsxtoe(i),plaatsytoe(i), plaatsztoe(i), richting(i))-
        (1-2*q(i)-2*(q(i))^2)/(1+q(i))^2*n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i))-
        (q(i))^2/(1+q(i))^2*n(plaatsxvorig(i),plaatsyvorig(i),plaatszvorig(i),richtingsterr(i));
    end

    %Applying the body force
    for i=1:19
        n(:,:,:,i) = n(:,:,:,i) + F(i);
    end
```

```
    %Applying the multireflection boundary condition
    for i=1:length(q)
        n(plaatsx(i), plaatsy(i), plaatsz(i), richtingsterr(i)) = BB(i);
    end

    %The propagation step
    n(:,:,:,2)=n(:,:,[zmax 1:zmax-1],2);
    n(:,:,:,3)=n(:,:,[2:zmax 1],3);
    n(:,:,:,4)=n(:,[ymax 1:ymax-1],:,4);
    n(:,:,:,5)=n(:,[2:ymax 1],:,5);
    n(:,:,:,6)=n([xmax 1:xmax-1],:,:,6);
    n(:,:,:,7)=n([2:xmax 1],:,:,7);
    n(:,:,:,8)= n([xmax 1:xmax-1],[ymax 1:ymax-1],:,8);
    n(:,:,:,9)= n([xmax 1:xmax-1],[2:ymax 1],:,9);
    n(:,:,:,10)=n([2:xmax 1],[ymax 1:ymax-1],:,10);
    n(:,:,:,11)=n([2:xmax 1],[2:ymax 1],:,11);
    n(:,:,:,12)=n([xmax 1:xmax-1],:,[zmax 1:zmax-1],12);
    n(:,:,:,13)=n([xmax 1:xmax-1],:,[2:zmax 1],13);
    n(:,:,:,14)=n([2:xmax 1],:,[zmax 1:zmax-1],14);
    n(:,:,:,15)=n([2:xmax 1],:,[2:zmax 1],15);
    n(:,:,:,16)=n(:,[ymax 1:ymax-1],[zmax 1:zmax-1],16);
    n(:,:,:,17)=n(:,[ymax 1:ymax-1],[2:zmax 1],17);
    n(:,:,:,18)=n(:,[2:ymax 1],[zmax 1:zmax-1],18);
    n(:,:,:,19)=n(:,[2:ymax 1],[2:zmax 1],19);

    %Calculating the Reynolds number
    Reynold = mean(mean(v_x(nx+r+2,:,:)))*D/vis;
end
```