FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Lubrication correction for simulating particle-fluid interaction with the lattice Boltzmann method

Markus Müller

Bachelor's Thesis

Lubrication correction for simulating particle-fluid interaction with the lattice Boltzmann method

Markus Müller

Bachelor's Thesis

Aufgabensteller:	Prof. Dr. Ulrich Rüde
Betreuer:	Dipl.-Ing. (FH) Dominik Bartuschat, M.Sc. (hons)
Bearbeitungszeitraum:	28.01.2013 – 28.05.2013

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 25. Mai 2013 .....

The lubrication force is an important problem in simulations of suspended particles with the lattice Boltzmann method. When two particles approach each other and get near contact, closer than the lattice spacing, the thin layer of fluid, still left between the particles, cannot be sufficiently simulated, which even results in the particles sticking together. In order to get realistic results, there are different means to calculate and apply lubrication forces to the particles. The goal of this thesis is to explain the derivation and implementation of a technique presented by Ding and Aidun: the ALD' method. As it follows a link-wise approach, the terms of force applied per link are derived. Another feature is that the gap between the particles near contact is treated in a way, which always leaves a layer of fluid in the discretised grid. This is explained and shown, as well. Furthermore the concept of virtual fluid is presented. The Chair for System Simulation already has developed a framework for these kinds of simulations, the widely applicable Lattice Boltzmann solver from Erlangen (waLBerla), thus the implementation only enhances the current code. Conclusively some comparisons between analytical estimations and results of special test cases are presented.

Contents

1 Introduction

Solid particles suspended in any kind of fluid are an ubiquitous phenomenon, often observed in nature. Erosion, sedimentation or segregation processes are only a few examples of these particulate flows. Simulating their behaviour correctly bears the challenge of implicating both, the interaction between the particles and the particle-fluid interaction. Especially when two particles get near contact, the occurring forces can be difficult to be captured. The spatial discretisation fails to resolve the behaviour of the thin layer of fluid. If particles approach each other closer than the lattice spacing it is most likely that there is no fluid taken into account, at all, as the solid objects are mapped to neighbouring cells in the grid. Since the repelling force is not taken into account, this results in the particles sticking together. In order to maintain physically correct behaviour, the lubrication correction has to be applied in the simulation.

Physically, the lubrication force is based on the inertia of the fluid. If two particles approach each other, the fluid between them has to be displaced. As the gap gets smaller, there is less space for the displacement and the necessary force for it increases. This results in a repelling force on the particles, getting stronger the faster they approach each other. If the objects separate, the opposite effect occurs. The increasing space between the particles has to be filled with the fluid which is more difficult with smaller gaps. This results in an attractive force on the objects that increases with the speed of the separating particles.

The first approach to incorporate these forces into the lattice Boltzmann simulation have been introduced by Ladd and Verberg. The idea is to calculate the forces by lubrication theory (as shown in section 2.5) based on a single link along the smallest gap between the particles. This results in the normal lubrication force, which is then applied to the objects. [Ladd and Verberg, 2001, p. 1232]

Ding and Aidun try to improve this first approach by introducing the so-called ALD' method. It follows a link-wise approach, i.e. the total force on a particle is a summation over smaller terms of force, calculated for certain links connecting the objects near contact. Furthermore, it includes a treatment of the discretised grid that always leaves a layer of fluid in between the objects, no matter how close they get. [Ding and Aidun, 2003, p. 694]

"The Chair for System Simulation (LSS) has developed a massively parallel and flexible simulation framework, which originally has been centered around the Lattice-Boltzmann method for the simulation of fluid scenarios. Meanwhile, its usability has been extended to a wide range of applications" [Chair for System Simulation, 2012], such as the simulation of large ensembles of suspended particles. Since it claims to be easily adaptable and a library for the development of new applications, as well, waLBerla will be the base of the implementation. As the given code will only be enhanced by the ALD' method, a basic knowledge of waLBerla is required to understand the implementation and the code excerpts shown in section 3.

The sections about the theory starts with a short introduction to the lattice Boltzmann method and compares the formulas stated by Ding and Aidun to those, on which the current implementation is based. After that the features of the ALD' method will be derived and explained, successively.

The second main part is the implementation. After an overview of the changes, code excerpts are shown and explained in details.

Conclusively two different scenarios for the validation of the implementation are presented. As there is an analytical estimation of the lubrication force, it is compared to the results of the simulations with both, the present approach and the ALD' method. Before the final conclusion, performance measurements of the sequential and parallel execution are shown.

2 Theory

2.1 Lattice Boltzmann Method

The lattice Boltzmann method is a microscopic kinetic based method for fluid simulation. Instead of directly solving the macroscopic variables, particle distribution functions, that describe clusters of particles at discrete positions moving with discrete velocities, are used. In the present implementation, the lattice Boltzmann method is applied to resolve the fluid-structure interaction. The Navier-Stokes equations, that describe the motion of Newtonian fluids, are not solved directly, but the discrete lattice Boltzmann equation is solved, as an alternative. Along with the other most important equations, this is presented in the notation used by Ding and Aidun.

The discrete velocities follow the directions of the D3Q19 model [Aidun et al., 1997, p. 3,Table 1]. They are represented by three-dimensional vectors, consisting only of zero or one valued entries which result in lengths of one, in off-diagonal direction, and $\sqrt{2}$, in diagonal direction. Each direction $e_{\sigma i}$ is specified by a unique combination of its subscripts. The inverse of this direction is stated as $e_{\sigma i'}$, with $e_{\sigma i} = -e_{\sigma i'}$.

The state of the fluid, at a specific node and time, is described by the particle distribution functions $f_{\sigma i}$. They are calculated through the lattice Boltzmann equation

$$f_{\sigma i}\left(x + e_{\sigma i}, t + 1\right) = f_{\sigma i}\left(x, t\right) - \frac{1}{\tau}\left[f_{\sigma i}\left(x, t\right) - f_{\sigma i}^{(0)}\left(x, t\right)\right] , \tag{2.1.1}$$

with τ being the relaxation scale and $f^{(0)}$ the equilibrium distribution function, stated as:

$$f_{\sigma i}^{(0)} = \rho\left(x\right)\left[A_\sigma + B_\sigma\left(e_{\sigma i} \cdot u\right) + C_\sigma\left(e_{\sigma i} \cdot u\right)^2 + D_\sigma u^2\right] . \tag{2.1.2}$$

The density ρ and the velocity u are defined as

$$\rho\left(x, t\right) = \sum_{\sigma, i} f_{\sigma i}\left(x, t\right) , \tag{2.1.3}$$

$$u\left(x, t\right) = \frac{1}{\rho\left(x, t\right)} \sum_{\sigma, i} f_{\sigma i}\left(x, t\right) e_{\sigma i} , \tag{2.1.4}$$

respectively. The coefficients A_σ, B_σ, C_σ and D_σ for the three-dimensional case are displayed in table 1. [Ding and Aidun, 2003, p. 687ff]

Table 1: Ding and Aidun equilibrium distribution function coefficients

center:	$A_0 = \frac{1}{3}$	$B_0 = 0$	$C_0 = 0$	$D_0 = -\frac{1}{2}$
off-diagonal:	$A_1 = \frac{1}{18}$	$B_1 = \frac{1}{6}$	$C_1 = \frac{1}{4}$	$D_1 = -\frac{1}{12}$
diagonal:	$A_2 = \frac{1}{36}$	$B_2 = \frac{1}{12}$	$C_2 = \frac{1}{8}$	$D_2 = -\frac{1}{24}$

In order to resolve the particle-fluid interaction the “link-bounce-back” scheme is used as a boundary condition. The particle distribution functions next to a moving solid surface are updated by

$$f_{\sigma i}\left(x, t + 1\right) = f_{\sigma i'}\left(x, t_+\right) + 2\rho B_\sigma u_b \cdot e_{\sigma i} , \tag{2.1.5}$$

with the link $e_{\sigma i'}$ pointing from a fluid cell to a node, covered by an object. The coefficient B_σ attains the values according to table 1 and u_b is the velocity of the solid particle. The time t_+ is immediately after the collision and $t_+ - t \ll 1$.

The resulting momentum that is transferred to the solid object is calculated via

$$\delta p_{\sigma i} = 2e_{\sigma i'} \left[f_{\sigma i} \left(x, t+1 \right) - \rho B_\sigma u_b e_{\sigma i} \right] , \tag{2.1.6}$$

using the distribution function, derived in equation (2.1.5). Note that σi still points away from the boundary node. [Ding and Aidun, 2003, p. 692]

2.2 Virtual Fluid

The implementation of waLBerla treats particles as real solid objects without fluid nodes inside them. The advantage of this approach is that the density of the particles can be lower than the fluid density. The disadvantage is that, as the particles move around in the domain, nodes change their state from being liquid to solid object and the other way round. The conversion to a solid object is fairly easy as only a flag needs to be set, to indicate that this node is left out in the lattice Boltzmann sweep. Converting a cell back to the liquid state needs some more work given that the node's values of the velocity, the density and the particle distribution functions are obsolete. Currently the particle distribution functions are being restored with the help of the equilibrium distribution function and the density becomes an average of the surrounding nodes' densities, like shown in figure 1.

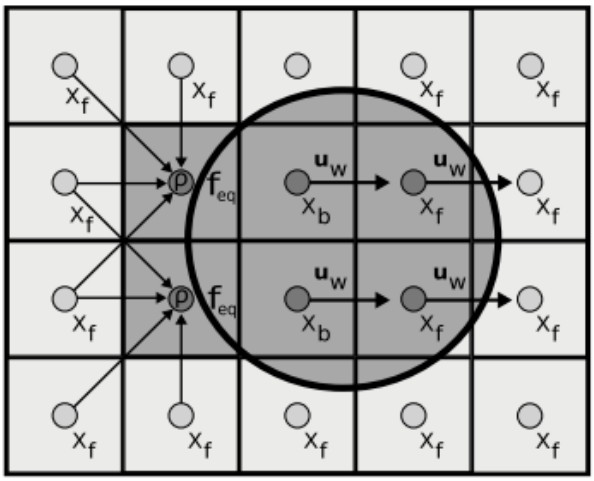


Figure 1: Restoring of uncovered nodes [Götz et al., 2010, p. 6]

Another approach to the depiction of suspended particles is the so called "Shell method". The entire domain is occupied by physical fluid and consequentially the force and torque of a particle are affected by both, the fluid nodes outside and those inside the solid objects boundaries. "The particles comprise a solid shell of given mass and inertia, filled with fluid of the same mass density as the bulk fluid. As a result, the total mass density of the particle can never be less than the fluid density, limiting this approach to cases where the solid-to-fluid density ratio is greater than one" [Ding and Aidun, 2003, p. 690]. The advantage is, of course, that the entire domain gets updated in every lattice Boltzmann sweep, so there is no need of restoring any values when the particles move.

The idea behind the virtual fluid is to combine the advantages of the two mentioned approaches. Instead of using physical fluid throughout the whole domain, that inside a solid particle becomes virtual. This way, the fluid nodes inside a solid particle are included in the lattice Boltzmann sweep, except that they do not contribute to the force and torque transferred to the object. By updating their densities, velocities and particle distribution functions in every time step, nothing needs to be restored if they happen to turn into physical fluid, i.e. get uncovered. Nevertheless, physical and virtual fluid are strictly separated. The contour of the solid object establishes a boundary. As a boundary condition the "link-bounce-back" scheme, already referred to in section 2.1, is used. Figure 2 shows the similarity of the treatment of physical and virtual fluid. The difference is, though, that the virtual fluid does not transfer any momentum to the solid particle by "bouncing" back off the object's contour. The updated values of the particle distribution functions are calculated via equation (2.1.5). [Ding and Aidun, 2003, p. 690ff]

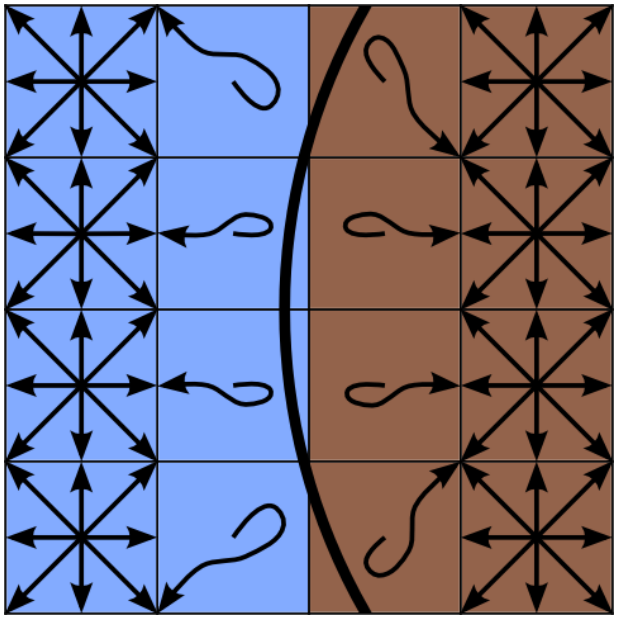
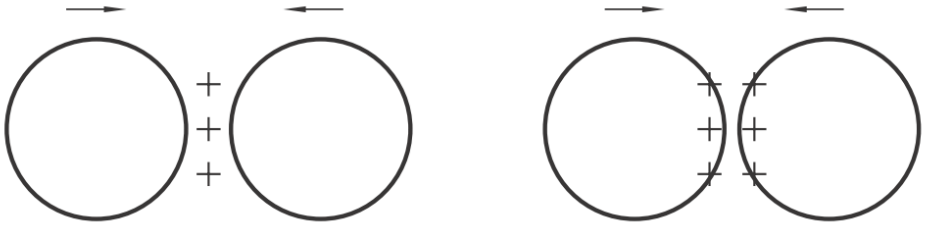


Figure 2: BC treatment of physical and virtual fluid

2.3 Particles near contact

If two particles approach each other and get very close, the hydrodynamic forces cannot be calculated correctly with the lattice Boltzmann method. When the gap between objects becomes smaller than a lattice unit, the cells mapped to the different particles neighbour directly, without a fluid node between them. In the exceptional case (figure 3 (a)), that the center of the node, which is responsible for its state, lies in the middle of the approaching objects, there will always be one fluid node, no matter how small the gap is. Without the fluid nodes there is no momentum transferred to the particle, of course, even though a small gap between the objects still exists, which, in reality, would be filled with the liquid. In order to improve the hydrodynamic interaction, the gap is artificially mapped by diminishing the particles. This holds for the exceptional case as well, since it is really rare and a single layer of fluid nodes is hardly enough to calculate the hydrodynamic forces correctly.

For the purpose of recognizing the cells in question, the term "bridge link" is defined. It stands for a direct connection of two nodes mapped to different particles, thus can be in both off-diagonal and diagonal direction. In case of the exceptional state, a link pointing to a fluid node, even though another particle is closer than a lattice unit, is referred to as "half-bridge". As the lubrication force is calculated link-wise



(a) At least one layer of fluid nodes in the gap

(b) No fluid nodes in the gap

Figure 3: Cases for approaching particles [Ding and Aidun, 2003, p. 692]

in the ALD' method, this definition plays an important role in section 3.3, as well. Figure 4 shows a few examples for bridge and half-bridge links.

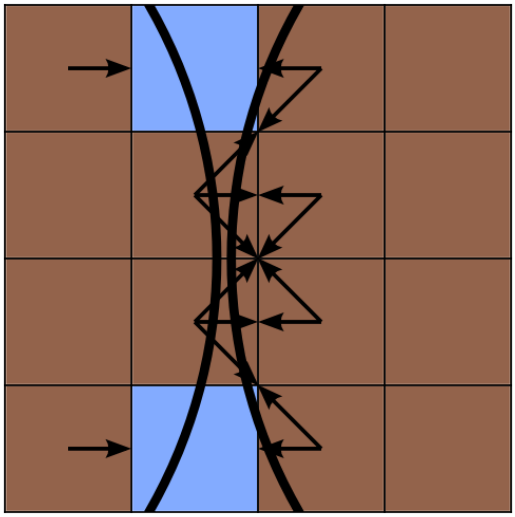


Figure 4: (Half-)Bridge links

In order to always keep the thin layer of fluid in a gap between two particles, nodes, actually mapping an object, are treated as physical fluid, like they were moved out of the particle. As all those cells near particles' contact contain at least one bridge link or half-bridge link, this feature is used as a condition to change the states of the nodes from virtual to physical fluid. The boundary treatment is influenced by this transition, too, as the borders move accordingly.

Figure 5 shows an example of two particles near contact. While on the left side there is no layer of fluid left to simulate the liquid in between the particles, the right side displays the state of the nodes after the explained transitions have been applied.

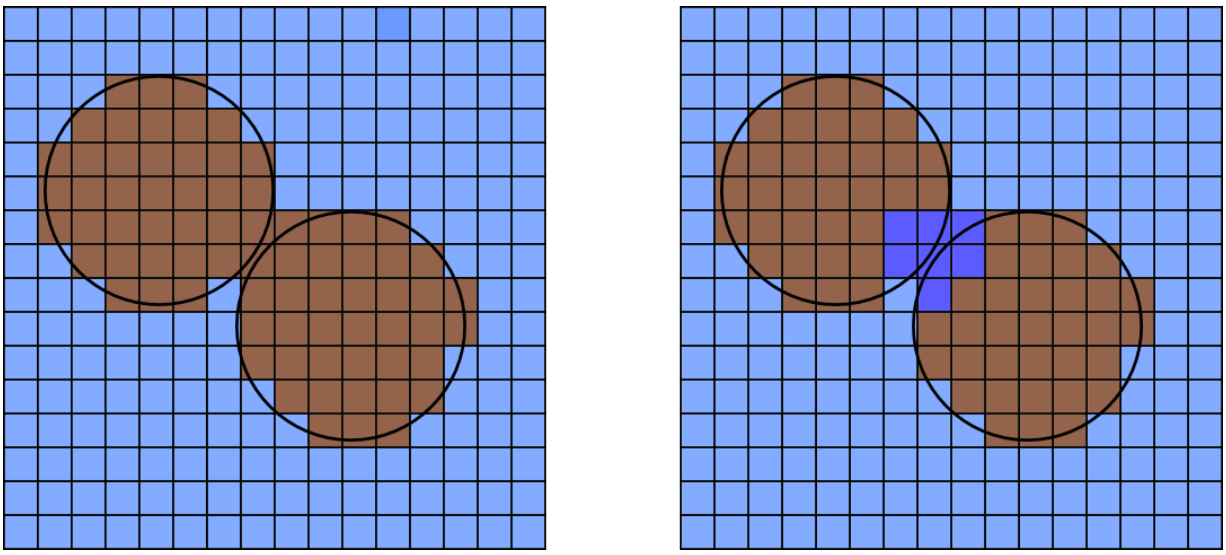


Figure 5: Treatment of nodes near particles' contact

2.4 Un-/Cover Force

As the particles move around in the domain, nodes get covered and uncovered. Figure 6 shows an example, where the particle moved from its former (dotted line) to its new position (solid line). By changing the state of a node, "a small impulse of force is applied to the solid particle" [Ding and Aidun, 2003, p. 690], caused by the difference of the cell's and the particle's velocity. The force on the object, if it covers a fluid node, is calculated through

$$F^{(c)}\left(x, t_0 + \frac{1}{2}\right) = \rho\left(x, t_0\right)\left[u\left(x, t_0\right) - U'\right] , \tag{2.4.1}$$

with U' being the local velocity of the particle. The formula, in case of an uncovered node, only differs by the sign:

$$F^{(c)}\left(x, t_0 + \frac{1}{2}\right) = -\rho\left(x, t_0\right)\left[u\left(x, t_0\right) - U'\right] . \tag{2.4.2}$$

Those forces are assumed to be distributed through a time step, represented by the interval $[t_0, t_0 + 1]$. As the particles move the total number of physical fluid nodes varies slightly, but on the macroscopic level this fluctuation is negligible. [Ding and Aidun, 2003, p. 690f]

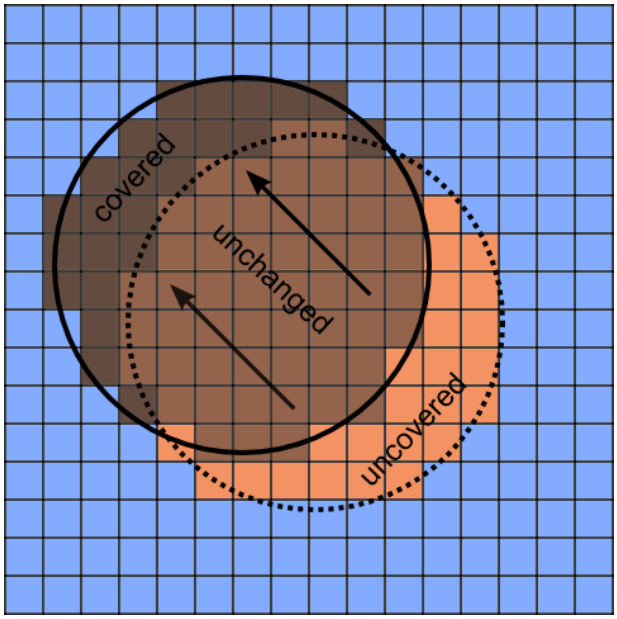


Figure 6: Nodes being un-/covered by the movement of a particle

2.5 Lubrication Force

As waLBerla implements the "relation given by Cox (1974) for two spheres" [Ding and Aidun, 2003, p. 696], the lubrication force is calculated through

$$F = -\frac{3\pi}{2\lambda^2}\frac{\nu\rho U}{\epsilon}\,, \tag{2.5.1}$$

with the kinematic viscosity ν and the surface curvature λ, given by

$$\lambda = \frac{1}{2}\left(\frac{1}{R_1} + \frac{1}{R_2}\right)\,. \tag{2.5.2}$$

The relative velocity U is determined by subtracting the objects velocity from that of the particle it approaches. This leads to a negative force - the objects repel each other - as the particles collide and, consequentially, to a positive force - they attract each other - when they separate. The surface curvature factor only applies for two spherical particles, the adaptation for a spherical object approaching a flat wall will be explained at the end of the section. The gap ϵ such as the radii of the two spheres R_1 and R_2 are shown in figure 7.

The problem using this relation is, that it only applies to cases where the surfaces of the spheres are connected by a single off-diagonal link. As the lubrication force gets only important the particles get very close, there are many connecting links that can be taken into account. Ding and Aidun extend the formula by excerpting an element of force, which, summed up over the surface elements, becomes the stated equation (2.5.1). It turns out to be

$$df' = \frac{3}{2}\frac{\nu\rho U}{\lambda\delta^2}\,, \tag{2.5.3}$$

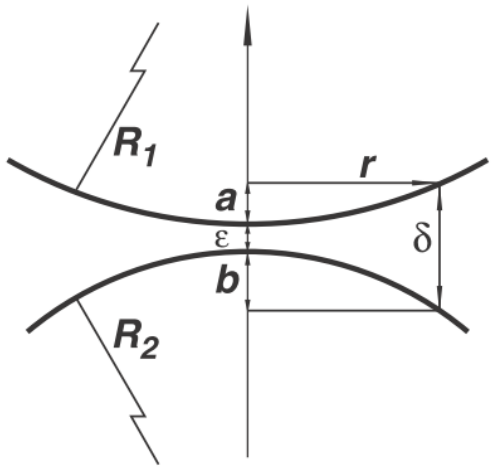


Figure 7: Notations used in the calculation of the lubrication force [Ding and Aidun, 2003, p. 695]

with δ being the surface separation, as shown in figure 7. In order to adapt the formulation to multiple links a weighting factor q is introduced. Therefore the general form of the element of force is given by

$$df = q\, df' = \frac{3}{2}\frac{\nu\rho U}{\lambda\delta^2} q\,. \tag{2.5.4}$$

The weighting factor depends on the number and the configuration of the connecting bridge and half-bridge links. It can be averaged as

$$\bar{q} = \frac{\sum q\, df'}{\sum df'}\,, \tag{2.5.5}$$

for which Ding and Aidun found in their simulations, "that the value of $\bar{q} \simeq 0.6$" [Ding and Aidun, 2003, p. 696]. As a hard coded value, this approximation does not only give good results but saves a lot of computational work, as well.

The next adaptation, that has to be considered, is that the connecting links can be in both, off-diagonal and diagonal direction. Since "it is necessary to keep isotropic and independent of the lattice structure" [Ding and Aidun, 2003, p. 697], the elements of lubrication force should be scaled according to the equilibrium distribution function. As shown in table 1, the off-diagonal coefficients are twice the size of the diagonal ones. Furthermore the lubrication force only gets relevant when the gap between the surfaces, in the direction of the connection, gets very small, i.e. smaller than the length of the link. Applying those derivations to equation (2.5.4) leads to the general form of the element of lubrication force

$$df = \begin{cases} \frac{3\bar{q}}{2c_\sigma^2\lambda}\nu\rho U\left(\frac{1}{\delta^2} - \frac{1}{c_\sigma^2}\right)\,, & \text{if } \delta < c_\sigma \\ 0\,, & \text{if } \delta \geq c_\sigma \end{cases} \tag{2.5.6}$$

Multiplying the element of force with the normalized directional vector of the connecting link gives

$$F^{(d)}(x,t) = df\frac{e_{\sigma i}}{c_\sigma}\,, \tag{2.5.7}$$

which is the term for the lubrication force on the solid particle. [Ding and Aidun, 2003, p. 695ff]

Applying equation (2.5.2) and equation (2.5.6) to this formula leads to the one, used to implement the ALD' method,

$$F = \frac{3\bar{q}\nu\rho U}{c_\sigma^3}\left(\frac{R_1 \cdot R_2}{R_1 + R_2}\right)\left(\frac{1}{\delta^2} - \frac{1}{c_\sigma^2}\right)e_{\sigma i}\,. \tag{2.5.8}$$

If a spherical particle approaches a flat wall, the formula can be adapted easily, by the object's radius $R_2 \mapsto \infty$, which leads to

$$F = \frac{3\bar{q}\nu\rho U}{c_\sigma^3} \cdot R_1 \cdot \left(\frac{1}{\delta^2} - \frac{1}{c_\sigma^2}\right)e_{\sigma i}\,. \tag{2.5.9}$$

It is necessary to point out, that all the physical values, used in the theory, are in lattice units! Given parameters in other units have to be transferred.

2.6 Comparison to implementation basis

The implementation of waLBerla follows a notation, which is, among others, used by Götz et al. [2010]. As the current code should only be enhanced, the consistency of the necessary formulas has to be checked before new features can be added.

The first and major difference is the depiction of the discrete directions. The subscripts σi are combined into a single factor α. This leads to $e_{\sigma i} = e_\alpha = -e_{\bar{\alpha}}$, with $e_{\bar{\alpha}}$ being the inverse direction.

Götz et al. "adopt a lattice Boltzmann collision scheme proposed by Bhatnagar, Gross and Krook" [Götz et al., 2010, p. 3]:

$$f_\alpha\left(x_i + e_\alpha, t + \triangle t\right) = f_\alpha\left(x_i, t\right) - \frac{1}{\tau}\left[f_\alpha\left(x_i, t\right) - f_\alpha^{(eq)}\left(x_i, t\right)\right]\,, \tag{2.6.1}$$

where $f^{(eq)}$ is the equilibrium distribution function. Given that σi and α represent the same direction and $\triangle t = 1$, equation (2.1.1) and equation (2.6.1) match, thus the used lattice-Boltzmann equations correspond.

The formulation of the equilibrium distribution function is stated as:

$$f_\alpha^{(eq)}\left(x_i, t\right) = \omega_\alpha\left[\rho\left(x_i, t\right) + \rho_0\left(3e_\alpha u\left(x_i, t\right) + \frac{9}{2}\left(e_\alpha u\left(x_i, t\right)\right)^2 - \frac{3}{2}u\left(x_i, t\right)^2\right)\right]\,, \tag{2.6.2}$$

with $\rho_0 = 1$ being defined. Applying this definition, the density and the velocity are calculated through equation (2.1.3) and equation (2.1.4), respectively. The weighting factors ω_α are "chosen as in Succi et al." [Götz et al., 2010, p. 3] and have the values of table 2 hard coded in waLBerla.

Table 2: Götz et al. equilibrium distribution function coefficients

center:	$\omega_0 = \frac{1}{3}$
off-diagonal:	$\omega_1 = \frac{1}{18}$
diagonal:	$\omega_2 = \frac{1}{36}$

In order to confirm the consistency of the weighting factors, it is advisable to reorganize equation (2.6.2). Keeping in mind that $\rho(x_i,t)=\rho_0=1$ leads to

$$f_\alpha^{(eq)}(x_i,t)=\omega_\alpha\cdot 1+3\omega_\alpha e_\alpha u(x_i,t)+\frac{9}{2}\omega_\alpha\left(e_\alpha u(x_i,t)\right)^2-\frac{3}{2}\omega_\alpha u(x_i,t)^2\ . \tag{2.6.3}$$

Comparing the just derived equation (2.6.3) to equation (2.1.2) results in the set-up of table 3. The inequalities in the comparison of coefficients B_0 and C_0 are never taken into account, as the directional vector in the center $e_0=0$ turns the expressions, containing the different weighting factors, to zero, anyway.

Table 3: Comparison of the equilibrium distribution function coefficients

$A_0=\frac{1}{3}=\omega_0$	$B_0=0\neq 3\omega_0=3\cdot\frac{1}{3}=1$	$C_0=0\neq\frac{9}{2}\omega_0=\frac{9}{2}\cdot\frac{1}{3}=\frac{3}{2}$	$D_0=-\frac{1}{2}=-\frac{3}{2}\omega_0=-\frac{3}{2}\cdot\frac{1}{3}$
$A_1=\frac{1}{18}=\omega_1$	$B_1=\frac{1}{6}=3\omega_1=3\cdot\frac{1}{18}$	$C_1=\frac{1}{4}=\frac{9}{2}\omega_1=\frac{9}{2}\cdot\frac{1}{18}$	$D_1=-\frac{1}{12}=-\frac{3}{2}\omega_1=-\frac{3}{2}\cdot\frac{1}{18}$
$A_2=\frac{1}{36}=\omega_2$	$B_2=\frac{1}{12}=3\omega_2=3\cdot\frac{1}{36}$	$C_2=\frac{1}{8}=\frac{9}{2}\omega_2=\frac{9}{2}\cdot\frac{1}{36}$	$D_2=-\frac{1}{24}=-\frac{3}{2}\omega_2=-\frac{3}{2}\cdot\frac{1}{36}$

The implementation of the "link-bounce-back" scheme follows

$$f_{\bar\alpha}(x_f,t)=\tilde f_\alpha(x_f,t)+6\omega_\alpha\rho_w e_{\bar\alpha}\cdot u_w\ , \tag{2.6.4}$$

with ρ_w being the density, close to the object, and u_w the particle's velocity. "$\tilde f_\alpha$ denotes the post-collision state of the distribution function" [Götz et al., 2010, p. 3], where α is the direction pointing to the boundary node. The values of the coefficients ω_α are stated in table 2.

Comparing the directions used in equation (2.1.5) and equation (2.6.4), one can see that this time $\sigma i'$ corresponds to α, as both point from a fluid to a boundary node. As $f(x,t_+)$ and $\tilde f(x,t)$, both, stand for the post-collision state of the same particle distribution function, these parts of the equations match as well. In order to confirm the consistency $2B_\sigma e_{\sigma i}=6\omega_\alpha e_{\bar\alpha}$, has to be valid. The values stated in table 3 confirm this condition, as the central directional vector $e_0=0$ in both cases.

The transferred momentum is calculated through

$$\delta p_\alpha=e_\alpha\left[2\tilde f_\alpha(x_f,t)+6\omega_\alpha\rho_w e_{\bar\alpha}\cdot u_w\right]\ , \tag{2.6.5}$$

with α still being the direction towards the solid object. In order to compare these two approaches, already knowing that $B_\sigma e_{\sigma i}=3\omega_\alpha e_{\bar\alpha}$, one has to bring them into a common direction. As described above, a directional vector can be turned into it's opposite direction by multiplying it by "-1". Applying this to δp_α and $\tilde f_\alpha(x_f,t)$ gives

$$-\delta p_{\bar\alpha}=e_\alpha\left[-2\tilde f_{\bar\alpha}(x_f,t)+6\omega_\alpha\rho_w e_{\bar\alpha}\cdot u_w\right]\ . \tag{2.6.6}$$

Comparing equation (2.6.6) to equation (2.1.6) shows, that not only the directions match but the signs correspond, as well.

As a result of the comparison, the consistency of both approaches is confirmed and therefore the already implemented functions of waLBerla stay in use for the ALD' method.

3 Implementation

Since the implementation of the ALD' method should extend waLBerla under the constraint, that the simpler lubrication correction stays functional, the existing functions (though some are slightly modified) are reused where possible. New created methods are written in a way, fitting the existing ones.

For the boundary condition handling of the virtual fluid, inside the particles, a new header file became necessary. "UBBBC_Virt" is just a copy of the standard "UBBBC", adapted to the virtual fluid. In order to use it, the "initFunctions" had to be expanded. "createVirtMovObstStdHandling" and "createVirtMOStandardHandling" imitate the behaviour of their existing namesakes. The combination of the flags LIQUID, MO and NEAR_OBST is replaced by VIRT_FLUID, VIRT_OBST and VIRT_NEAR_OBST, respectively.

"setWallRelation" is another addition of the initialisation, to set the relations of the walls to the relation field, according to their positions.

A third new "initFunction", "correctVirtFluidInit", got necessary, as the boundary condition handling of the virtual fluid resulted in the whole domain being initialised with the virtual fluid flag. The method removes this, wrongly set, flag and additionally sets the VIRT_OBST flag to all cells containing LIQUID.

A new LBM Sweep function, "LBM_MO_AidunVirt_SweepOptim", is added, too. The only change is the adaptation to the virtual fluid by the use of the virtual source field, from which the particle distribution functions of the virtual fluid are read.

The majority of the modifications has been done in the mapping. A new sweep, the "obstacleMappingAidun", has been created, imitating the existing mapping sweep. Its functionality will be explained in the following sections.

3.1 Mapping the flags

In waLBerla a flag field is provided to set the states of the nodes. The flags used to implement the ALD' method are:

- LIQUID
  marks all cells that map physical fluid
- MO
  marks the nodes covered by moving obstacles, i.e. the particles
- NEAR_OBST
  marks fluid cells that neighbour a node with a MO flag, i.e. are next to a boundary
- OLD
  marks all cells that have been MO in the last time step
- VIRT_FLUID
  marks all cells that contain virtual fluid, i.e. are covered by an object; even though MO marks the exact same nodes, this flag is necessary because of the bc handling of the virtual fluid
- VIRT_NEAR_OBST
  marks all cells containing virtual fluid that neighbour a LIQUID node, i.e. are next to a boundary for the virtual fluid
- VIRT_OBST
  marks all cells that are obstacles to the virtual fluid, i.e. cells containing physical fluid

- BRIDGE_LINK
  marks all cells that contain at least one bridge or half-bridge link
- NO_SLIP or FREE_SLIP
  mark fixed boundaries, such as walls

Initially, all cells are set to LIQUID. After that, the fixed objects are marked, according to their boundary behaviour, with either NO_SLIP or FREE_SLIP. In each time step, algorithm 1, written in a pseudo code, is applied. Since the particles move by one node, at most, the flags only change in an area of two cells wider than the particles' contour. In order to avoid unnecessary work, all the mapping functions are called with a bounding box as parameters, which includes all the cells in question.

The first step is the reinitialisation. It removes the flags from the last time step, to prevent them from affecting the setting of the new ones. Furthermore, it sets the OLD flag to all cells that have been covered by a particle, i.e. had the MO flag set. After that, the obstacles are mapped by setting the MO flag to the covered cells. Now that the positions of the obstacles are known, the occurrence of half-bridge links or bridge links can be checked. If either of them is found in a cell, the BRIDGE_LINK flag has to be set. As each of theses cells is treated as physical fluid, the flags have to be changed from MO to LIQUID. To complete the transition, the relation has to be set to zero. This is done after the determination, because it could be influenced by this change of the state. Now, that only cells which are treated as a particle contain the MO flag, the VIRT_FLUID flag is added to those nodes. The VIRT_OBST flag is set to all cells containing the LIQUID flag, accordingly, as it is necessary for the boundary condition handling. The last step of the mapping is the setting of the NEAR_OBST and VIRT_NEAR_OBST flags. Of course this has to wait until transitions from particle to fluid, and the other way round, are correctly set, but it can be combined with the setting of the VIRT_FLUID flag.

Algorithm 1: Mapping sweep

```
1 for ( all moving obstacles )
2     reInit();
3     setCells();
4     setBridgeLinks();
5     correctBridgeLinkFlags();
6     setVirtualFluid();
7     setNearObst();
```

For the purpose of determining whether a bridge or half-bridge link exists, the relation field is taken into account. Since every object in waLBerla has a unique ID assigned to it, they are mapped to the cells in order to easily access by which object the nodes are covered. In case of a fluid cell the relation is set to zero. To determine the bridge links, algorithm 2 is applied to every MO cell of the given bounding box. The determination, if the neighbouring cell is covered by a different particle than the current one, is done by a comparison of the "IDs", stored in the "relation" field. The calculated distance is the one, that separates the two particles in the given direction. As it is necessary for the lubrication force, the details will be explained in section 3.3. If the directly neighbouring node is not mapped to a particle, i.e. its relation is zero, then the next further cell has to be checked, as well. The exceptional case, with one fluid node between the particles even though their distance is small enough to fulfil the criteria for a half-bridge link, refers to section 2.3 and is shown in figure 4. Its handling is done in the last "else if" statement.

Figure 8 shows an example of two particles near contact and the correct mapping of the flags.

Algorithm 2: Bridge link determination

```
for ( D3Q19::iterator dir = D3Q19::beginNoCenter(); dir != D3Q19::end(); ++dir )
    targetID = relation->GET_SCALAR(pos+dir);
    if ( targetID != 0 && targetID != objectID )
        delta = calcDistance();
        if ( delta < dir.length() )
            flags->GET_SCALAR(pos) |= BRIDGE_LINK;
    else if ( (targetID == 0) && ((flags->GET_SCALAR(pos+dir) & GHOST) == 0) )
        newTargetID = relation->GET_SCALAR(pos+2*dir);
        if ( newTargetID != 0 && newTargetID) != objectID )
            delta = calcDistance();
            if ( delta < dir.length() )
                flags->GET_SCALAR(pos) |= BRIDGE_LINK;
    else if ( (targetID == 0) && ((flags->GET_SCALAR(pos+dir) & GHOST) == GHOST) )
        for ( all objects )
            if ( objIt->containsPoint(offset+pos+2*dir) )
                delta = calcDistance();
                if ( delta < dir.length() )
                    flags->GET_SCALAR(pos) |= BRIDGE_LINK;
                    break;
```

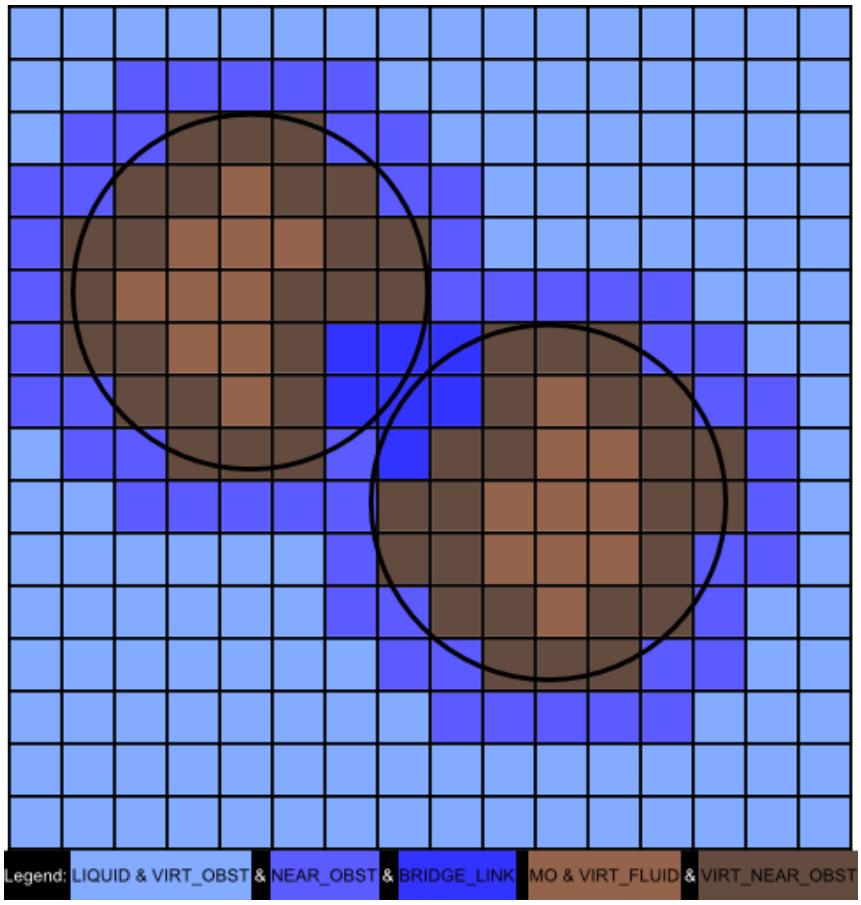


Figure 8: Example of correctly mapped flags

3.2 Un-/Cover Force

The calculation of the force, caused by covering and uncovering a fluid node with a solid particle, follows equation (2.4.1) and equation (2.4.2), accordingly. The implementation applies algorithm 3, with "pos1" being the center of the node. The velocity field stores the fluid velocities and is accessed by "velocity->GET_SCALAR()". The indices zero to two represent the x-, y- and z-values, respectively. The velocity of the solid particle is accessed by the function "velFromWF(pos1)", where "pos1" is the center of the node. As usual in the ALD' method, the density in lattice units is chosen for the calculation. Since the sign is the only difference in the named equations, the computation of the absolute value can be made independently of the case. In order to determine if a node has been covered or uncovered, the flags are considered. As shown in algorithm 1, every cell that contained the MO-flag in the last time step got marked with the OLD-flag. If there is a cell containing the MO-flag but not the OLD-flag, it has to have been fluid and just got covered by a particle. In this case the positive force is added to the particle. Should the cell be LIQUID and furthermore contain the OLD-flag, it has been a particle and thus just got uncovered. Now, the negative value of the force has to be applied to the particle. Moreover, the relation has to be set to zero, i.e. marking the cell as fluid and the OLD-flag has to be removed, as they are only set by the "reInit"-function shown in algorithm 1. In case of a cell containing both, the MO-flag and the OLD-flag, it has to be removed, as well, but since this cell has been a solid particle and stays one, it is neither covered nor uncovered and thus no force is applied.

Algorithm 3: Un-/Cover force

```
1  // getting velocity from field
2  const pe::Vector3<real_t> vel = (relation->GET_SCALAR(x,y,z))->velFromWF(pos1);
3  // calculating the un-/cover force
4  pe::Vec3 coverForce(velocity->GET_SCALAR(x,y,z)[0] - vel[0],
       velocity->GET_SCALAR(x,y,z)[1] - vel[1], velocity->GET_SCALAR(x,y,z)[2] -
       vel[2]);
5  coverForce *= lbmData->rho_L();
6
7  // treat covered nodes
8  if ( (flags->GET_SCALAR(x,y,z) & (MO | OLD)) == MO ) {
9      relation->GET_SCALAR(x,y,z)->addForceAtPos(coverForce, pos1);
10 }
11 // treat uncovered nodes
12 if ( (flags->GET_SCALAR(x,y,z) & (LIQUID | OLD)) == (LIQUID | OLD) ) {
13     relation->GET_SCALAR(x,y,z)->addForceAtPos(-1.0*coverForce, pos1);
14     relation->GET_SCALAR(x,y,z) = 0;
15     flags->GET_SCALAR(x,y,z) -= (flags->GET_SCALAR(x,y,z) & OLD);
16 }
```

3.3 Lubrication Force

Before the lubrication force, following equation (2.5.8), can be calculated, the distance of the particles along a specific link has to be determined. Considering the triangle, set up by the center of the node, the center of the spherical particle and the intersection of the link with the contour of the sphere, to be the sum of two vectors leads to

$$\left|\left(n \cdot \vec{dir}\right) + \left(\vec{x} - \vec{posSphere}\right)\right| = radius\,, \tag{3.3.1}$$

with x being the center of the node, *dir* being the vector of the discrete direction and *posSphere* being the center of the sphere. Resolving n leads to a quadratic equation, as there is a second intersection point in the opposite direction. The given direction *dir* already has the correct sign, so that only the positive solution of n needs to be calculated. This way, the distances inside of the two spheres near contact can be determined, accordingly, and subtracting them from the distance between the centres of the two nodes gives the gap, necessary for the calculation of the lubrication force.

The implementation is shown in algorithm 4, with "SQR(x)" being x^2 and "sqrt(x)" being $\sqrt{x}$. The indices zero to two represent the x-, y- and z-values of the vectors. "dir.cx()" gives the x-value of the direction which is either zero or one, according to the D3Q19 model. The y- and z-values can be accessed, correspondingly.

Algorithm 4: Distance calculation sphere-sphere

```
1  // calculating factors for the quadratic equation of object sphere
2  const real_t oA = SQR(dir.cx()) + SQR(dir.cy()) + SQR(dir.cz());
3  const real_t oB = 2*x[0]*dir.cx() - 2*posOSphere[0]*dir.cx() + 2*x[1]*dir.cy() -
       2*posOSphere[1]*dir.cy() + 2*x[2]*dir.cz() - 2*posOSphere[2]*dir.cz();
4  const real_t oC = SQR(x[0]) - 2*x[0]*posOSphere[0] + SQR(posOSphere[0]) +
       SQR(x[1]) - 2*x[1]*posOSphere[1] + SQR(posOSphere[1]) + SQR(x[2]) -
       2*x[2]*posOSphere[2] + SQR(posOSphere[2]) - SQR(oSphere->getRadius());
5
6  // calculating relative distance in object sphere
7  const real_t n = (-oB + sqrt(SQR(oB) - 4*oA*oC)) / (2*oA);
8
9  // reverting direction for target sphere
10 stencil::D3Q19::iterator inv(dir.inverseDir());
11
12 // calculating factors for the quadratic equation of target sphere
13 const real_t tA = SQR(inv.cx()) + SQR(inv.cy()) + SQR(inv.cz());
14 const real_t tB = 2*y[0]*inv.cx() - 2*posTSphere[0]*inv.cx() + 2*y[1]*inv.cy() -
       2*posTSphere[1]*inv.cy() + 2*y[2]*inv.cz() - 2*posTSphere[2]*inv.cz();
15 const real_t tC = SQR(y[0]) - 2*y[0]*posTSphere[0] + SQR(posTSphere[0]) +
       SQR(y[1]) - 2*y[1]*posTSphere[1] + SQR(posTSphere[1]) + SQR(y[2]) -
       2*y[2]*posTSphere[2] + SQR(posTSphere[2]) - SQR(tSphere->getRadius());
16
17 // calculating relative distance in target sphere
18 const real_t m = (-tB + sqrt(SQR(tB) - 4*tA*tC)) / (2*tA);
19
20 // calculating the distance between the centers of the object and target cell
21 const real_t cellDistance = sqrt(SQR(y[0]-x[0]) + SQR(y[1]-x[1]) +
       SQR(y[2]-x[2]));
22
23 // calculating the gap between spheres in the given direction
24 delta = cellDistance - n*dir.length() - m*inv.length();
```

This distance is also required for the determination of the bridge-links, as already mentioned in section 2.3. In order to do as few computationally expensive root calculations as possible, the lubrication force is included in the sweep that sets the bridge links, shown in algorithm 2.

Equation (2.5.8) is applied to compute the actual value of the lubrication force and implemented, as shown in algorithm 5. As usual in the ALD' method, the density and the viscosity are, both, converted to lattice units. The relative velocity U is calculated by subtraction of the object's velocity from the target particle's velocity. This way, it is negative if the particles collide, so the direction of the force is the inverse of the links direction, i.e. the particles repel each other. Should the particles separate, the signs change, of course, and the object attracts the target particle. In order to get the part of the velocity that is in direction of the particles near contact, it is multiplied with the normalized vector, connecting the centres of the objects.

Algorithm 5: Lubrication force

```
1  // setting constants
2  const real_t q = 0.6;
3  const real_t nu = lbmData->nu_L();
4  const real_t rho = lbmData->rho_L();
5  // calculating relative velocity of the particles
6  pe::Vec3 velDiff( targetID->getLinearVel() - objectID->getLinearVel() );
7  real_t U = trans( velDiff ) * (tSphere->getPosition() -
       oSphere->getPosition()).getNormalized();
8  // calculating the lubrication force
9  real_t lubF  = ( (3.0*q*nu*rho*U) / (dir.length()*dir.length()*dir.length()) ) *
       ((oSphere->getRadius()*tSphere->getRadius()) /
       (oSphere->getRadius()+tSphere->getRadius())) * ((1.0/SQR(delta)) -
       (1.0/SQR(dir.length())));
10 // adding the lubrication force to the particle
11 objectID->addForceAtPos(lubF*dir.cx(), lubF*dir.cy(), lubF*dir.cz(), x[0], x[1],
       x[2] );
```

4 Validation

4.1 Analytical Estimation

In order to validate the implemented functions, an analytical estimation of the expected results is necessary. The force, taken into account, is the normal force on the sphere. According to Claeys and Brady, the "leading order force, f, based on lubrication approximation between two spheres in three-dimensional space is given by" [Ding and Aidun, 2003, p. 702]

$$\frac{f}{2\rho\nu U/\lambda} = \frac{3\pi}{4s} + C_w \,, \tag{4.1.1}$$

with $C_w \approx 0$ being a constant, depending on the wall effect. The separation value s is calculated through

$$s = \epsilon \cdot \lambda \,, \tag{4.1.2}$$

with ϵ being the gap between the spheres. This way, the variable parameters of the force are cancelled out and the function can be used for various scenarios.

Resolving the force leads to

$$f = \frac{3\pi}{4 \cdot \frac{\epsilon}{R}} \cdot (2\rho\nu UR) = \frac{3}{2}\pi\rho\nu UR^2 \cdot \frac{1}{\epsilon} \,, \tag{4.1.3}$$

with ϵ being the only variable parameter during the test.

If only one sphere approaching a flat wall is considered, the function can be adapted easily by applying the change in λ, explained in section 2.5, and gets to

$$f = \frac{3\pi}{4 \cdot \frac{\epsilon}{2R}} \cdot (4\rho\nu UR) = 6\pi\rho\nu UR^2 \cdot \frac{1}{\epsilon} \,, \tag{4.1.4}$$

with ϵ, now, being the gap between the sphere and the wall.

4.2 Parameters

The parameters, used in the tests, have to fulfil certain criteria in order to stay consistent. For the validation, the length of the lattice unit $\triangle x = 10^{-3}\,\mathrm{m}$, the magnitude of the time step $\triangle t = 0.3\,\mathrm{s}$, the kinematic viscosity, in lattice units, $\nu = 0.3$ and the density $\rho_s = 10^3 \frac{\mathrm{kg}}{\mathrm{m}^3}$, which gives $\rho = 1$ in lattice units. The kinematic viscosity, based on physical quantities, can be calculated via

$$\nu = \nu_s \cdot \frac{\triangle t}{\triangle x^2}\,, \tag{4.2.1}$$

which results in $\nu_s = 10^{-6} \frac{\mathrm{m}^2}{\mathrm{s}}$. The relaxation time constant is computed via

$$\tau = \frac{6 \cdot \nu + 1}{2}\,, \tag{4.2.2}$$

which results in $\tau = 1.4$. Another criterion is that the Reynolds number $Re = Re_s$.

$$Re = \frac{u \cdot d}{\nu}\,, \tag{4.2.3}$$

with u being the lattice scale velocity and d being the characteristic length of the blunt body which results in

$$\frac{u \cdot d}{\nu} = \frac{u_s \cdot d}{\nu_s}\,. \tag{4.2.4}$$

As a sphere is used in the tests, $d = 2R$ with $R = 8.5$. Furthermore, a fixed constant velocity $u = 0.02$ is set, which leads to $Re = 1.13$.

In case of different given parameters, the stated equations have to be fulfilled and can be used to calculate the unknown ones. [Ding and Aidun, 2003, p. 700]

4.3 Sphere-Sphere Test

The scenario for the "Sphere-Sphere" test is a three-dimensional channel, being periodic in x-direction. In the middle of the channel two spheres approach each other with a fixed velocity of $u_{1/2} = \pm 0.01$, which leads to a relative velocity of $U = 0.02$. The analytical estimation is therefore calculated, according to equation (4.1.3) and with the given parameters, as

$$f = \frac{3}{2} \cdot \pi \cdot 1 \cdot 0.3 \cdot 0.02 \cdot 8.5^2 \cdot \frac{1}{\epsilon} \approx \frac{2.0428}{\epsilon}\,, \tag{4.3.1}$$

depending only on ϵ, which is the gap between the two spheres. The measured results are shown in figure 9.

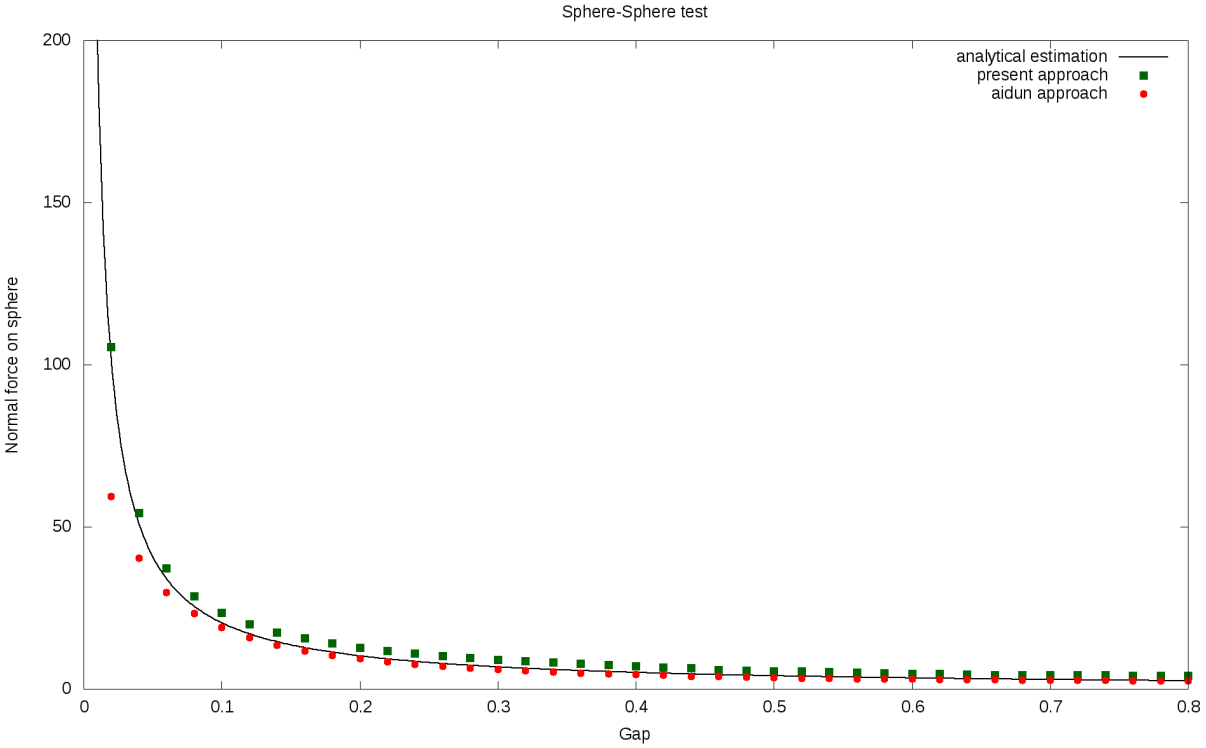


Figure 9: Results of the Sphere-Sphere test

Analysing the results, it is obvious that both approaches follow the characteristics of the analytical values. For gaps bigger than 0.1, the ALD' method matches the estimation almost exactly and differs only by about 8%, in average. If the gap gets closer than that, the measured values tend to become too small. Since the spheres are positioned in the middle of the channel, the centres always move, in x-direction, along a grid line. This way, the smallest gap lies exactly between the shortest off-diagonal bridge links. Because of the curvature of the spheres, the gaps, considered for the calculation, are bigger, which results in smaller lubrication forces. In the opposite case, with an off-diagonal bridge link being the shortest gap, the values grow too strong. In an arbitrary scenario, the values will be somewhere in between these extremes and match the analytical estimation.

The forces on the second sphere are negative, of course, but as they have identical absolute values it is not necessary to explicitly show them.

4.4 Sphere-Wall Test

In the "Sphere-Wall" test there is a three-dimensional channel, as well. Instead of a second sphere, a fixed wall seals the channel at the lower end of the x-direction. The velocity of the sphere is, again, set to the constant value of $U = 0.01$, which equals the relative velocity, as the wall doesn't move. According to equation (4.1.4), the analytical estimation is calculated via

$$f = 6 \cdot \pi \cdot 1 \cdot 0.3 \cdot 0.01 \cdot 8.5^2 \cdot \frac{1}{\epsilon} \approx \frac{4.0856}{\epsilon}, \tag{4.4.1}$$

which is exactly two times the value of the "Sphere-Sphere" test. The measured values are shown in figure 10.

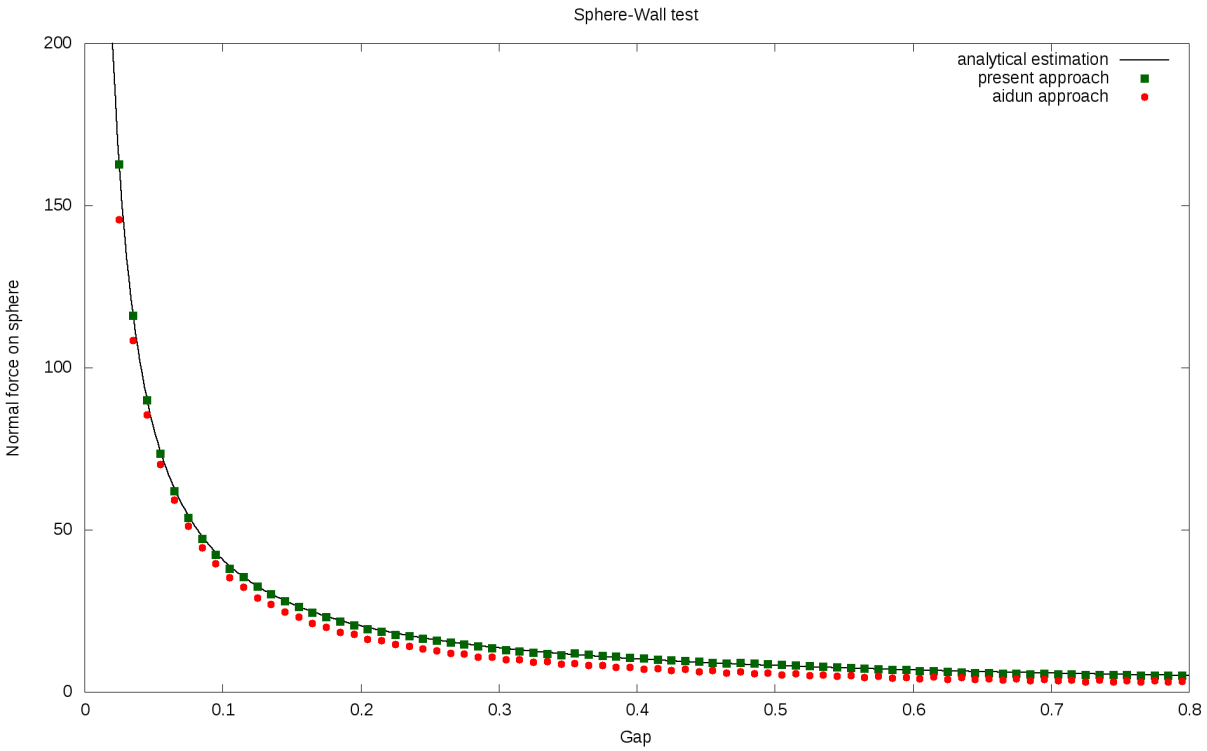


Figure 10: Results of the Sphere-Wall test

The results of the present approach match the analytical estimation almost perfectly. The measured values of the ALD' method differ up to about 30% from the estimation, for bigger gaps. The less space between the particles is left the smaller gets the relative error. For gaps smaller than 0.1 the effect of the surface curvature increasing the distance along the bridge links, explained in section 4.3, has to be taken into account.

4.5 Performance

In order to evaluate the performance of the code, the overall duration of the execution of a test is measured, as well as the time spent for the mapping and the calculation of the lubrication force. In the following, they will be referred to as execution and calculation time, respectively.

The execution of the code can be parallelised by partitioning of the domain. Each block of cells is then handled by a specific process. It depends on the hardware, how many processes can run simultaneously. The measurements have been carried out on an Intel Core i7-2600 processor with four physical cores and hyper-threading, which results in eight virtual cores. Since the calculation of the lubrication force is performed for each particle, the parallelisation takes only effect if the objects are distributed over different processes. The calculation time will always be the longest duration needed by a single process.

The sequential execution of the "Sphere-Sphere" test with the present approach takes about $55s$. The more complex mapping and calculation of the ALD' method prolong this value to $87s$. Even more notable is the difference of the calculation times, which double from $0.30s$ to $0.60s$. Since the mapping, as well as the calculation of the lubrication force, are more complex in the ALD' method, these considerably prolonged durations had to be expected.

Running the test with the ALD' method on eight processes, separating the domain into four blocks in x-direction and two blocks in y-direction, results in an execution time of about $39s$. A speed-up by a factor of 2.26 seems quite good, since the parallelisation causes a considerable amount of additional computational work as well as data transfers between the processes. The calculation time is shortened to $0.36s$, i.e. reduced by a factor of 1.7. When the domain is split into blocks a ghost layer has to be added to handle the transition from one process to another. Because of these additional cells, the speed-up factor has to be less than two.

For the "Sphere-Wall" test the size of the domain is doubled in each direction, i.e. eight times the total size. The sequential execution of the ALD' method takes $617s$, which is a factor of 7.1 longer than the "Sphere-Sphere" test. As there is only one particle and no periodicity, the growth of the domain is not directly transferred to the execution time. For the same reason, the calculation time is even decreased from $0.60s$ to $0.41s$.

Applying the same parallelisation as in the "Sphere-Sphere" test leads to an execution time of $386s$. Since there is only one particle, the speed-up is decreased by a factor of 1.6. For this reason and the fact, that the parallelisation increases the computational work, the duration of the calculation grows slightly from $0.41s$ to $0.47s$. Comparing these times to the results of the present approach, which only takes about $0.17s$, shows the increased complexity of the ALD' method, again.

These results confirm the expected effects of the parallelisation. For scenarios with more, evenly distributed, particles the speed-up will be even better.

5 Conclusion

The validation shows that the results of the implementation are very close to those of the single link approach, in terms of the normal force. Having proven that the ALD' method works for the simple scenarios, used in section 4, it seems to be a promising alternative. In more complex tests, the multiple link approach is very likely to give more accurate results, but the determination of the normal force is not implemented for those cases, yet. Furthermore, an adaptation of the distance and force calculation to tubes, additionally to spheres and walls, might be useful to solve current problems in simulations. This is an interesting task for future work, as it makes a new approach in terms of the surface curvature necessary.

The whole implementation of the ALD' method (including the mapping, the calculation of the distance and the forces) is way more complex than the present calculation of the lubrication force. The results of the validation tests show, that the execution times almost double. Even though the scaling of the parallelisation seems to be good, the calculation times are too big. Only a huge improvement of the results could justify these prolongations, but they almost match those of the current approach. Improving the speed of the implementation by optimising the code might be a task for future work, as well.

Although it is not explicitly shown in a test case, the tangential part of the lubrication force is too large, if calculated by the ALD' method. The way it is implemented now, it is not usable for physically correct simulations. A possible workaround is to restrict the forces to the normal direction, as the validation proved them to be correct. Since the normal forces can be applied according to analytically correct formulas, the inclusion of the tangential forces would have been a major advantage of the ALD' method. Loosing this benefit makes the increased complexity of the calculations even more disadvantageous. Nevertheless, implementing the workaround, or even correcting the error in the method, might be possible tasks for future work, again, and are necessary in order to use the method in waLBerla.

All in all, the ALD' method seems like the first step in a promising direction, but there are still a lot of problems to be solved.

References

C.K. Aidun and J.R. Clausen. Lattice-Boltzmann Method for Complex Flows. *Annual Review of Fluid Mechanics 2010*, pages 439–472, 2010.

C.K. Aidun and Y. Lu. Lattice Boltzmann Simulation fo Solid Particles Suspended in Fluid. *Journal of Statistical Physics*, pages 49–61, 1995.

C.K. Aidun, Y. Lu, and E.-J. Ding. Dynamic Simulation fo Particles Suspended in Fluid. *IPST technical paper series*, 1997.

C.K. Aidun, Y. Lu, and E.-J. Ding. Direct analysis of particulate suspensions with inertia using the discrete Boltzmann equation. *J. Fluid Mech.*, pages 287–311, 1998.

Chair for System Simulation. waLBerla. http://www10.informatik.uni-erlangen.de/en/Research/Projects/walberla/, 2012.

Chair for System Simulation. waLBerla - Description. http://www10.informatik.uni-erlangen.de/en/Research/Projects/walberla/description.shtml, 2013.

I.L. Claeys and J.F. Brady. Lubrication singularities of the grand resistance tensor for two arbitrary particles. *Physicochem. Hydrodyn.*, pages 261–293, 1989.

E.-J. Ding and C.K. Aidun. Extension fo the Lattice-Boltzmann Method for Direct Simulation fo Suspended Particles Near Contact. *Journal of Statistical Physics*, pages 685–708, 2003.

Christian Feichtinger. *Design and Performance Evaluation of a Software Framework for Multi-Physics Simulations on Heterogeneous Supercomputers*. PhD thesis, Friedrich-Alexander Universität Erlangen-Nürnberg, 2012.

J. Götz, K. Iglberger, C. Feichtinger, S. Donath, and U. Rüde. Coupling Multibody Dynamics and Computational Fluid Dynamics on 8 192 Processor Cores. *Parallel Computing*, pages 142–151, 2010.

Matthias Hofmann. Parallelisation of Swimmer Models for the Simulation of Swarms of Bacteria in the Physics Engine pe. Master thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013.

A.J.C. Ladd and R. Verberg. Lattice-Boltzmann Simulations of Particle-Fluid Suspensions. *Journal of Statistical Physics*, pages 1191–1251, 2001.

Balthasar Reuter. Particle Suspension and Lubrication. Presentation, 2012.

Jorge Israel Bernal Romero. Fluid-Particle Interaction with the Immersed Boundary Lattice Boltzmann Method. Master thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2012.

K. Yeo and M.R. Maxey. Simulation of concentrated suspensions using the force-coupling method. *Journal of Computational Physics*, page 2401–2421, 2010.