Report: Lattice Boltzmann

Derrick Timmerman S3523799, Wesley Seubring S3529797, Group 06

October 26, 2018

Abstract

Lattice Boltzmann method is relatively new, in the field of simulating fluid dynamics. Lattice Boltzmann method is widely used for simulating flows.

Firstly this research will present the basic theory of the lattice Boltzmann method. Secondly the paper will present the correlation between the Reynolds number and the turbulence of a single fluid simulation using the Lattice Boltzmann method. The simulation simulates a single fluid that has an outflow to the right with a single cylinder in the center. The result of the simulation shows how increasing the Reynolds number increases the turbulence, when looking at the velocity of the fluid.

Contents

1 Introduction

Lattice Boltzmann method (LBM) is a very commonly used method for the simulation of fluids. It can work with multi-phase elements, calculate the flow through complex geometries, and allows for the addition of calculation complex processes as a heat transfer in a flow [11]. When implementing LBM, many lattice models are available for simulating components in a dimensional space [14]. In addition, intensive research in more complex simulations of LBM has already been done, including the simulation of multiple components, multiple phases and heat transfers. [17, 8] .

The first part of the research will cover the theory of LBM. The second part discusses and analyses the flow of a single fluid in a LBM simulation with respect to the Reynolds number. Reynolds number is used as a metric for the viscosity of a fluid [7]. The experiment will show how a change in the Reynolds number is correlated to the turbulence.

In section 2, the methods and theory relevant to the experiment are discussed. In section 3, the experiments of this research will be described. The results are shown in section 4 and discussed in section 5. In section 6, the results will be summarized.

2 Theory

2.1 Lattice Boltzmann Method

The Lattice Boltzmann method (LBM) is a numerical approach on the mechanic of fluid simulation. LBM looks at macroscopic properties of a node and the microscopic properties of the particles in a node to compute the behaviour of a fluid. The implementation of the LBM is relatively easy and can make great use of parallel computing. On top of LBM there are plenty of models that extend on the LBM to compute more complex problems like multi-component flows and fluid simulation in complex geometries (e.g. the flow in arteries) [2, 10].

This chapter will explain the basic theory of the Lattice Boltzmann method. Starting with some key equations for LBM.

In earlier research of Eduard Boltzmann, he described statistical mechanics in the Bolzmann equation to predict microscopic (atoms and molecules) and macroscopic (matter) properties [1]. In the Boltzmann equation is the collision term Ω, the rate between the particles, defined as follows:

$$\frac{\partial f}{\partial t} + \vec{c} \cdot \vec{\nabla} f = \Omega \tag{1}$$

where ∂t is a time increment and c is a vector of velocities. The equation is a distributions function with a source term Ω.

This collision term is very complicated and the Lattice Boltzmann method uses a discretized form of the simplified collision term, that was introduced by

Bhatnagar, Gross and Krook (BGK) [7]. They used the following collision term:

$$\Omega = \omega(f^{eq} - f) = \frac{1}{\tau}(f^{eq} - f) \tag{2}$$

where $\omega = \frac{1}{\tau}$. The ω is the collision frequency and τ is the relaxation factor. f^{eq} is the local equilibrium distribution function, for which Maxwell-Boltzmann distribution is an expansion (See Equation 3). This discretized equation can be used for finite dimensions of a grid in LBM.

LBM equations can be derived from a Navier-Strokes equation. The Navier-strokes equation is a mathematical description of the motion of a viscous fluid. Part of LBM refers to this method to describe fluid motion.[11]

The general algorithm of LBM is defined as follows:

1. First the scale of the system is changed from a physical system to the LBM system. Therefore we have to set the velocity of u_0 by using lattice velocity and a **lattice model** to define the velocity weights ω.
2. Initialize the **distributions functions**. The initial velocities are set using the lattice model as well as the initial density ρ.
3. Impose the chosen **boundary conditions** at macroscopic level and get the distribution of the streaming.
4. Compute the **collision** between nodes in the same region [7]).
5. Compute the **streaming** in all directions, satisfying the boundary conditions.
6. Repeat steps 3 to 5 until the stopping condition is reached.

2.2 Lattice models

This paragraph shows the two-dimensional lattice models that are used in the LBM. The common notation for these models is DxQy where x stands for the number of dimensions, and where y refers to the number of discrete velocity directions or linkages that each node has [1]. Some typical two-dimensional models can be seen in Figure 1. Each velocity direction, with respect to the central node, is provided with a weighting factor ω_k. These weighting factors are important for calculating the equilibrium distribution function, as described in subsection 2.3. The values of these weights depend on the distance between the particular node and the central node. The weighting factors ω_k for D2Q4, D2Q5 and D2Q9 are shown in Table 1. Here column 'rest' refers to a lattice velocity direction with $f_0 = 0$, column 'slow' refers to a lattice velocity moving on only one axis (x, y or z), and column 'fast' refers to a lattice velocity which moves along two axis. Combining Figure 1 and Table 1, the weighting factor for each velocity can be determined. For example, nodes f_1,f_2,f_3 and f_4 would be labeled as the 'slow' ones. The normalization condition for the weighting factors can be formulated as follows:

$$\sum_k \omega_k = 1$$

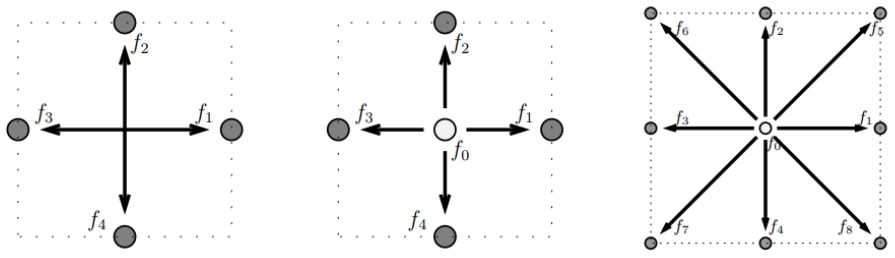

Figure 1: Two-dimensional models (left to right): D2Q4, D2Q5, D2Q9 [7]

Model	ω_k(rest)	ω_k(slow)	ω_k(fast)
D2Q9	$\frac{4}{9}$	$\frac{1}{9}$	$\frac{1}{36}$
D2Q5	$\frac{2}{6}$	$\frac{1}{6}$	-
D2Q4	-	$\frac{1}{4}$	-

Table 1: Weighting factors D2Q4, D2Q5 and D2Q9

2.2.1 Two-dimensional models

For simulations of two-dimensional flows, the D2Q9 model is most frequently used [7]. Although D2Q9 requires more computational effort compared to more simplified two-dimensional models, such as D2Q4 or D2Q5, the results are significantly more accurate [7]. Besides, simplified models as D2Q4 or D2Q5 are not suitable for simulating flow as they lack in lattice symmetry. This prevents the lattice Boltzmann equation from finding the appropriate Navier-Stokes equation, as it cannot be recovered from inappropriate lattices [15, 1].

2.3 Equilibrium distribution function

The equilibrium distribution function f^{eq} is an expansion of the Maxwell-Boltzmann distribution for low Mach number M. The normalized Maxwell-Boltzmann distribution function is as follows:

$$f = \frac{\rho}{2\pi/3} \exp^{-\frac{3}{2}(c-u)^2} = \frac{\rho}{2\pi/3} \exp^{-\frac{3}{2}(c\cdot c)} \exp^{\frac{3}{2}(2c\cdot u - u\cdot u)} \qquad (3)$$

where u is the velocity of the particles, c are the velocity vectors for the specified lattice model and ρ is the mass density. Next, we expand Equation 3.

$$f = \frac{\rho}{2\pi/3} \exp^{-\frac{3}{2}(c\cdot c)}[1 + 3(3(c \cdot u)) - \frac{3}{2}(u \cdot u) + \frac{9}{2}(c \cdot u)^2] \tag{4}$$

The equation is of the second order so it matches the order of the Navier-Stokes equations. Now we hide the non-linearity of the collision term. The equilibrium distribution function is of the following form:

$$f_i^{eq} = \rho\omega_i[1 + 3(3(c_i \cdot u)) - \frac{3}{2}(u \cdot u) + \frac{9}{2}(c_i \cdot u)^2] \tag{5}$$

where i runs from 0 to M (Mach number) and w_i is the weight of the lattice models for all the directions. The full deriving of the equation can be found in the paper by Igor Mele [7].

2.4 Collision step

In the collision step we compute the relaxation towards the local equilibrium whereby colliding particles are being resolved. The collision step makes use of the discretized BGK collision term (see Equation 5) [7].

$$f_i(x_i, l + \Delta l) = f_i(x_i, l) - \frac{\Delta l}{\tau}(f_i - f_i^{eq}) \tag{6}$$

where x is the lattice spacing and Δl is time step. So for a given time step the distribution is calculated using discretized BGK collision term.

2.5 Streaming step

In the streaming step the particles stream over to neighbours according to their lattice velocities. As seen in Equation 6 the streaming uses the LHS of the collision step for computing the streaming step [7].

$$f_i(x_i, l + e_i\Delta l, l + \Delta l) = f_i(x_i, l + \Delta l)) \tag{7}$$

where e_i is a space step. So, this computes a distribution according to the movements in all directions of the particles.

2.6 Boundary conditions

In computational fluid dynamics, boundary conditions are an important part and therefore it is necessary to apply techniques that mimic these boundary conditions. The behaviour of several different types of boundary conditions for the LBM are widely researched topics [13]. Boundary conditions that are very precise are not only required for simulations with complex geometry, but also for simulations with simple geometry like a cylinder in this research.

There are several types of boundary conditions but this research will limit itself to the following boundary conditions used in fluid simulations:

1. Periodic boundary conditions
2. Bounce-back boundary conditions
   (a) Full way bounce-back boundary conditions
   (b) Half way bounce-back boundary conditions

2.6.1 Periodic boundary condition (PBC)

This is the simplest type of boundary conditions and is often used in large systems for simulating an infinite or a continuous flow. When implementing PBC's, edges within the simulation environment act as if they are attached to the opposite side [12]. This means that nodes on the edge of the system, stream some of its particles to nodes connected to the opposite edge of the system [10]. This results in nodes passing through one edge, and re-appear on the opposite side with the same velocity, as can be seen in Figure 2.

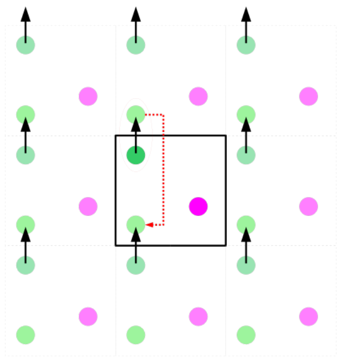

Figure 2: Illustration of periodic boundary conditions within the black square. Nodes crossing the edge are being replaced on the opposite side [12].

2.6.2 Bounce-back boundary conditions

For moving boundary conditions, non-slip conditions, flow over obstacles, or for modelling solid stationary, bounce-back boundary conditions are often used [7]. As the name suggests, with this technique particles moving towards a solid boundary will bounce back into the flow domain [1]. Bounce-back boundaries consist of a distribution function bounce-back scheme used at walls and objects to obtain a no-slip velocity condition ($u = 0$). To let the nodes bounce back from a solid, we have to recalculate the distribution function f_k. Taking D2Q9 in account (Figure 1), this can easily be done with the next equation:

$$f_2 = f_4,\ f_5 = f_7,\ f_6 = f_8 \tag{8}$$

What this equation actually does, is bouncing particles back to the node from which they were originally streamed which causes their direction of motion to be reversed. There are two slightly different schemes that are recommended for implementing bounce-back conditions. In the following paragraphs we are using the D2Q9 model as an example to explain these schemes in more detail.

2.6.2.1 Full way bounce-back boundary conditions

The scheme for full way bounce-back suggests placing a solid on the nodes itself. This can be seen in Figure 3. Here Ω represents the fluid and $\partial\Omega$ represents the boundary. In the streaming step a fluid population leaves the fluid node and reaches the boundary solid node. Afterwards, 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.

One thing to keep in mind, during previous research, it has been noticed that the full way bounce-back scheme gives first order spatial accuracy in boundaries [3, 4, 5].

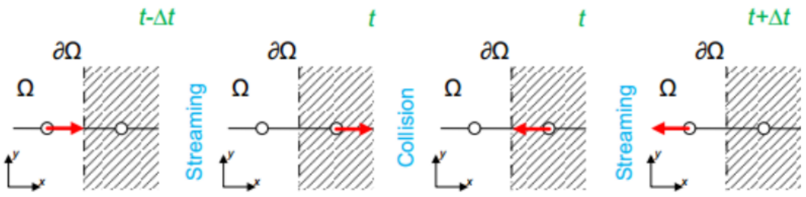

Figure 3: Full way bounce-back. Source: [16].

2.6.2.2 Half way bounce-back boundary conditions

The difference between half way and full way bounce-back is the fact that the scheme for half way bounce-back suggests placing a solid in the middle of the lattice nodes instead of on the nodes itself as can be seen in Figure 4. 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 between the fluid node and the solid node, at time $t + \frac{\triangle t}{2}$. Here the direction of streaming reverses. The fluid population moves back to the fluid node in the streaming step, having the reversed direction of motion.

The differences compared to full way bounce-back are a decrease in time steps (1 vs. 2), and an increase in accuracy as half way bounce-back is second order in numerical accuracy [3, 4, 5].

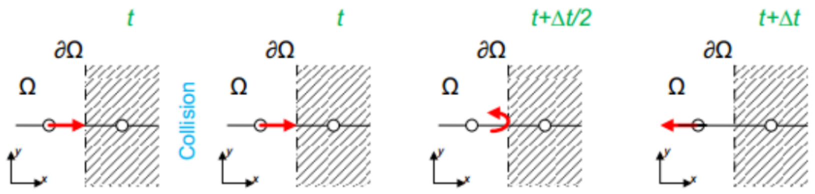

Figure 4: Half way bounce-back. Source: [16].

2.7 Reynolds number

The Reynolds Number, referred to as Re, is a convenient parameter to determine if a flow condition will be laminar or turbulent. When viscous forces are dominant (slow flow and low value for Re), it can be interpreted that they are sufficient to keep the fluid particles in line. In this case the flow is laminar. When the fluid flows faster in combination with a larger value for Re, we can interpret that the inertial forces dominate, and that the flow is turbulent. To compute Reynolds number for a LBM space the following equation can be used:

$$Re = \frac{U_{lbm} N}{v_{lbm}} \tag{9}$$

where U_{lbm} is the time step in LBM, v_{lbm} is the velocity in lattice units and N is the number of nodes along the direction [7].

The critical Reynolds number for flows:

Flow type	**Reynolds Number Range**
Laminar	up to $Re = 2300$
Transition	$2300 < Re < 4000$
Turbulence	$Re > 4000$

Table 2: Flow types and their corresponding Reynolds Numbers [6]

3 Experiments

In previous chapters the theoretical background of LBM has been discussed, it is now possible to perform experiments based on this theory. As this research is about simulating fluids around a cylinder, this experiment started by setting up a simulation model using the D2Q9 model with full-way bounce-back conditions. The implementation (section 8) is written in Python and is an altered version of the available on Palabos [9].

3.1 Reynolds number

The goal of the experiment is to observe flow patterns in different fluid viscosity's and see the correlation between the Reynolds number used in the simulation and the turbulence. As the Reynolds number represents the viscosity of the fluid, different flow patterns should occur by adjusting the Reynolds number. Therefore, by inputting a lower value for the Reynolds number the expectation is to see a more laminar flow, while at high Reynolds number an increase of the turbulence is expected. The following values for Reynolds number are used in this experiment:

$$Re = 1,\ \ Re = 220, Re = 1000, Re = 3000, Re = 4500$$

4 Results

For each Reynolds number, a simulation is performed whereby the state of the flow is captured after running 2000 iterations. These images should give an impression about the fluid flow for different Reynolds numbers. The results can be seen in the following figures:

Re	Figure
1	Figure 5
220	Figure 6
1000	Figure 7
3000	Figure 8
4500	Figure 9

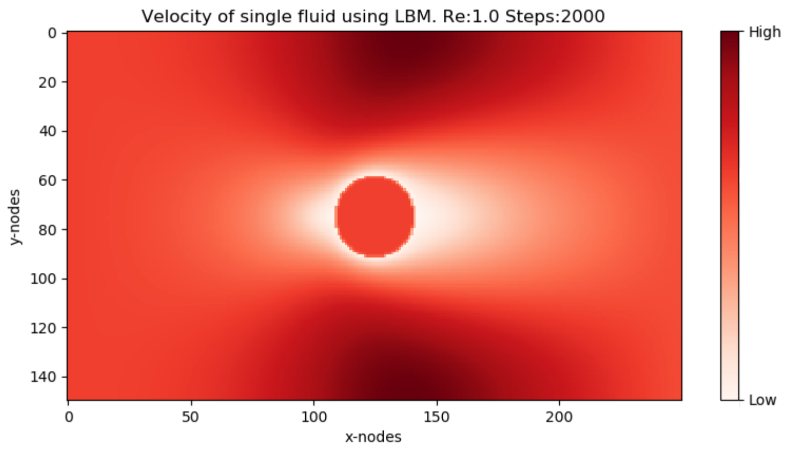

Figure 5: $Re = 1$

This image shows the flow of a fluid where $Re = 1$. You can see that the fluid has a constant speed until the fluid hits the cylinder in the center of the image. Left of the cylinder you see that velocity of the fluid decreases rapidly. Also you can see that the fluid tries to move past the cylinder, since both below and above the cylinder the speed increased. After the cylinder, the fluid starts to return to its original speed.

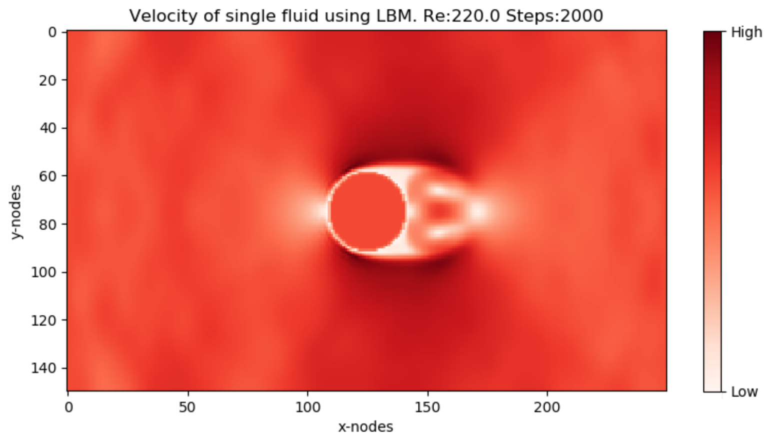

Figure 6: $Re = 220$

This image shows $Re = 220$ where you can see a circular flow emerging after the cylinder. In addition, the speed is more aligned to the cylinder.

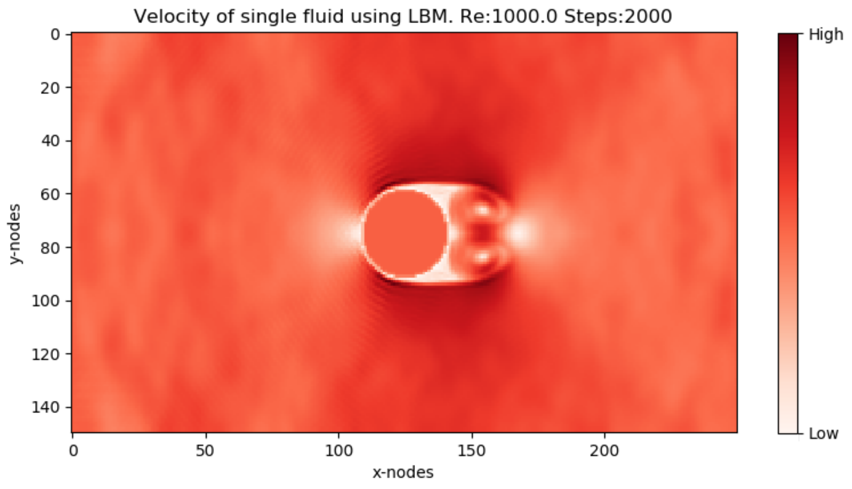

Figure 7: $Re = 1000$

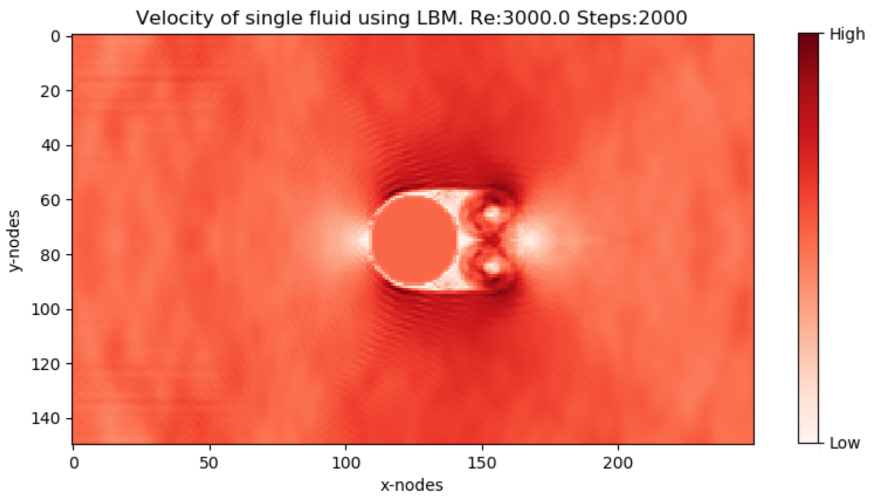

Figure 8: $Re = 3000$

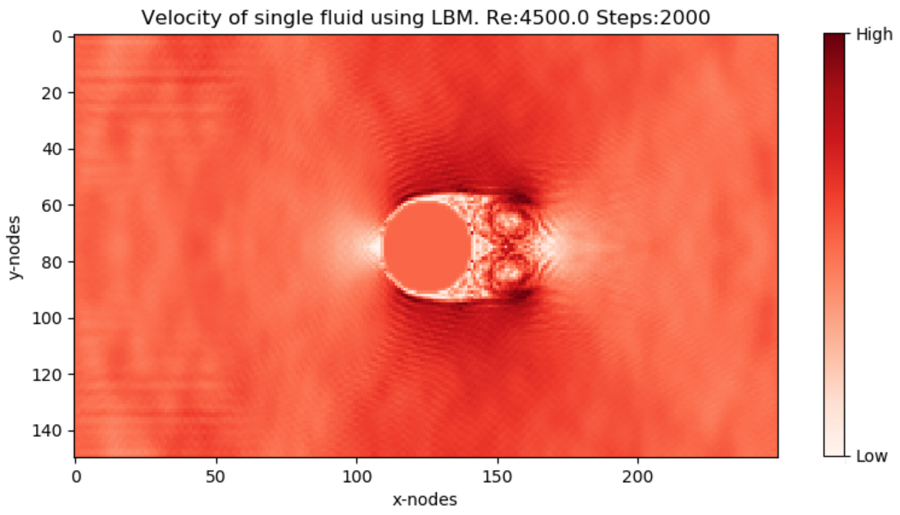

Figure 9: $Re = 4500$

5 Discussion

Comparing the results of $Re = 220$ with $Re = 1$ (see Figure 5 and Figure 6), it can be observed that the flow is still mostly laminar with minimal signs of turbulence around the cylinder. This is as expected as a Reynolds number of 220 is still very low. With $Re = 1000$ the turbulence becomes slightly more visible (see Figure 7), in the range 140 - 160 of the x-nodes the circulating flow behind the cylinder start to fluctuate slightly in velocity. With $Re = 3000$ the flow leaves the laminar phase and moves towards the transition phase, which can be observed in Figure 8 within the range 140 - 160 of the x-nodes. This figure shows a further increased fluctuation of the velocity behind the cylinder, to the point where it becomes difficult to see individual circulation flows. This figure also shows the first signs of chaotic behaviour by the appearance of small parts with high velocities close to the cylinder. With $Re = 4500$ in Figure 9 we observe real chaotic behaviour as the fluid does not follow a straight pattern anymore, but is affected heavily by multiple velocities behind the cylinder. Referencing to subsection 2.7, this situation is typical for a turbulent flow.

Considering above results, a correlation can be observed between the Reynolds number and the pattern of the fluid flow. An increasing value for Reynolds number leads to an increasing turbulence.

6 Conclusion

This research tried to give a good understanding of the general LBM, by describing its core theory. This knowledge is used for an experiment regarding the correlation between Reynolds number and the turbulence in a single fluid simulation. The outcome of this experiment is mostly in line with the expectations. The results of the experiment visually show the correlation between the Reynolds number and the turbulence in the simulation.

7 Work load

All the work was done together, so the work load was split evenly.

References

[1] A. A. Mohamad. *Lattice Boltzmann Method.* 2011.

[2] Meskas, J. Bill, Bao, Y. *Lattice Boltzmann Method for Fluid Simulations.* 2011. URL: `https://www.math.nyu.edu/~billbao/report930.pdf`.

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

[4] J. G. Georgiadis R. O. Buckius D. R. Noble S. Chen. "An evaluation of the bounce-back boundary condition for lattice boltzmann simulations". In: *International journal for numerical methods in fluids* 25 (249 1997).

[5] D.P. Ziegler. "Boundary conditions for lattice Boltzmann simulations". In: *J. Stat. Phys.* 71 (1993), pp. 1171–1177.

[6] Frank White. *Fluid Mechanics.* 4th edition. 2002.

[7] Igor Mele. *Lattice Boltzmann method.* 2003.

[8] Zhixin, L. Jinku, W. Moran, W. *A lattice Boltzmann algorithm for fluid–solid conjugate heat transfer.* 2006.

[9] *Lattice Boltzmann in various languages.* Accessed on: 11-10-2018. URL: `http://wiki.palabos.org/numerics:codes`.

[10] E. Magnus Viggen. *The lattice Boltzmann method: Fundamentals and acoustics.* 2014. URL: `https://www.researchgate.net/profile/Zoltan_Horvat/post/What_is_lattice_boltzmann_methodLBM_all_about/attachment/59d630ed79197b807798eb98/AS%5C%3A363442914512896%5C%401463662951192/download/Erlend+Magnus+Viggen+-+The+lattice+Boltzmann+method.pdf`.

[11] Peng, C. *The Lattice Boltzmann Method for Fluid Dynamics: Theory and Applications.*

[12] *Periodic boundary conditions.* Accessed on 21-10-2018. URL: `https://en.wikipedia.org/wiki/Periodic_boundary_conditions`.

[13] Li-Shi Luo Pierre Lallemand. *Lattice Boltzmann method for moving boundaries.* 2002.

[14] Lallemand, P. Qian, Y. H. D'humiéres, D. *Lattice BGK Models for Navier-Stokes Equation.* 1992.

[15] S. Succi. "The lattice Boltzmann Equation for Fluid Dynamics and Beyond". In: *Oxford University Press* (2001).

[16] S.C. Wetstein. *IMPLEMENTING THE LATTICE-BOLTZMANN METHOD.* 2014.

[17] Hudong, C. Xiaowen, S. *Lattice Boltzmann model for simulating flows with multiple phases and components.* 1992.

8 Appendix

Code Listing 1: D2Q9 LBM simulation code

```
from numpy import *;
from numpy.linalg import *
import matplotlib.pyplot as plt;
from matplotlib import cm

#https://en.wikipedia.org/wiki/Reynolds_number
# low == more smooth fluid | High == more chaotic and ↩
    turbulant flow
n_iterations = 20000  # Total number of time iterations.
Reynolds_number = 220.0
water_density = 1.0

plot_width = 400
plot_height = 400

ly = plot_height - 1.0 #TBD


# number of lattice velocities
q = 9

#Cylinder coordinates
cylinder_x = plot_width / 2
cylinder_y = plot_height / 2
cylinder_radius = plot_height / 9

#Velocity in lattice units
velocity_lattice_units = 0.04

#todo Define vars
nulb = velocity_lattice_units * cylinder_radius / ↩
    Reynolds_number
relaxation_time = 1.0 / (3. * nulb + 0.5)  # Relaxation ↩
    parameter.

###### Lattice Constants ######
D2Q9_FAST_WEIGHTING_VECTOR = 1. / 36.
D2Q9_SLOW_WEIGHTING_VECTOR = 1. / 9.
D2Q9_REST_WEIGHTING_VECTOR = 4. / 9.
D2Q9_SPEED = 1. / 3.

lattice_directions = array([(x, y) for x in [0, -1, 1] for ↩
    y in [0, -1, 1]])

#Intit all weights to fast
```

```python
lattice_weights = D2Q9_FAST_WEIGHTING_VECTOR * ones(q)

#Init the xaxis and yaxis horizontal with slow weights
lattice_weights[asarray([norm(direction) == 1. for direction in lattice_directions])] = D2Q9_SLOW_WEIGHTING_VECTOR;

#Set center particle to a rest weight
lattice_weights[0] = D2Q9_REST_WEIGHTING_VECTOR

#Find al the inverse indexes of the directions of the lattice #aka: noslip
inverse_direction_indexes = []
for direction in lattice_directions:
    inverse_direction_index = lattice_directions.tolist().index((-direction).tolist())
    inverse_direction_indexes.append(inverse_direction_index)

#Unknow indexes to the walls. todo: Find a nice way to find the indexes of lattice_directions
index_range = arange(q)
left_direction_indexes = index_range[asarray([direction[0] < 0 for direction in lattice_directions])]
vertical_direction_indexes = index_range[asarray([direction[0] == 0 for direction in lattice_directions])]
right_direction_indexes = index_range[asarray([direction[0] > 0 for direction in lattice_directions])]

sumpop = lambda fin: sum(fin, axis=0)  # Helper function for density computation.

def equilibrium(density, velocity):  # Equilibrium distribution function.
    # To find out: why multiplying with magic number 3.0
    # Calculates velocity for each direction of each element (e.g. 2x2 plane results in 9x2x2 arrays)
    direction_velocities = 3.0 * dot(lattice_directions, velocity.transpose(1, 0, 2))

    # Calculating squared velocity by the D2Q9_SPEED property
    squared_velocity = (velocity[0] ** 2 + velocity[1] ** 2) / (2 * D2Q9_SPEED)

    # Initialising zero-matrices for each direction
    equilibrium_distribution = zeros((q, plot_width, plot_height))

    # Equilibrium mass distribution
    for i in range(q): equilibrium_distribution[i, :, :] = density * lattice_weights[i] * (1. + direction_velocities[i] + 0.5 * direction_velocities[i] ** 2 - squared_velocity)
    return equilibrium_distribution

###### Setup: cylindrical obstacle and velocity inlet with perturbation ########
obstacle = fromfunction(lambda x, y: (x - cylinder_x) ** 2 + (y - cylinder_y) ** 2 < cylinder_radius ** 2, (plot_width, plot_height))
lattice_velocities = fromfunction(lambda d, x, y: (1 - d) * velocity_lattice_units * (1.0 + 1e-4 * sin(y / ly * 2 * pi)), (2, plot_width, plot_height))
equilibrium_distribution = equilibrium(water_density, lattice_velocities)
fin = equilibrium_distribution.copy()

###### Main time loop ############################################################

for time in range(n_iterations):
    #fin[left_direction_indexes, -1, :] = fin[left_direction_indexes, -2, :]  # Right wall: outflow condition.
    density = sumpop(fin)  # Calculate macroscopic density and velocity.
    velocity = dot(lattice_directions.transpose(), fin.transpose((1, 0, 2))) / density

    velocity[:, 0, :] = lattice_velocities[:, 0, :]  # Left wall: compute density from known populations.
    density[0, :] = 1. / (1. - velocity[0, 0, :]) * (sumpop(fin[vertical_direction_indexes, 0, :]) + 2. * sumpop(fin[left_direction_indexes, 0, :]))

    equilibrium_distribution = equilibrium(density, velocity)
    #fin[right_direction_indexes, 0, :] = fin[left_direction_indexes, 0, :] + equilibrium_distribution[right_direction_indexes, 0, :] - fin[left_direction_indexes, 0, :]
    fin[right_direction_indexes, 0, :] = equilibrium_distribution[right_direction_indexes, 0, :]
    fout = fin - relaxation_time * (fin - equilibrium_distribution)  # Collision step.
    for i in range(q): fout[i, obstacle] = fin[inverse_direction_indexes[i], obstacle]
    for i in range(q):  # Streaming step.
```

```
        fin[i, :, :] = roll(roll(fout[i, :, :], lattice_directions[i, 0], axis=0), lattice_directions[i, 1], axis=1)

    #   if (time % 100 == 0):  # Visualization
    plt.clf();
    plt.imshow(sqrt(velocity[0] ** 2 + velocity[1] ** 2).transpose(), cmap=cm.Reds)
    plt.pause(0.001)
```