The Lattice Boltzmann model

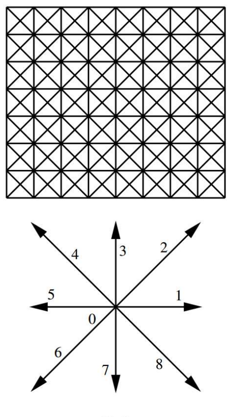
A short report by: Tom de Krom (4281594) and Vincent Heusinkveld (4390725)

Introduction

In this project, the Lattice Boltzmann model is considered. It is a so-called 'toy model', solely based on the laws of energy, mass and momentum conservation. And it uses the isotropic relaxation of stress.

The Lattice Boltzmann model

In the model, only densities of particles, n_i, and velocity, v_i, with a certain direction are taken into account. These directions correspond to the vectors labeled I making up the lattice. In this case, a 9 vector lattice is used called the d2q9 lattice as shown in the figure below (Thijssen 2007):



d2q9

It seems that the weights in Thijssen 2007 for the vectors of length sqrt(2) are mixed up with those of length 1. This can be seen in, for example, Bao and Meskas 2011, and other sources. Hence we reversed them in our code.

The algorithm

The algorithm works as follows (based on Thijssen 2007):

- The densities n_i are moved to the appropriate neighbour according to the direction of the lattice vectors.
- The velocities are calulated at each point and are weighted according to their respective density.
- If the velocities pass over the boundaries they should be reversed.
- A small velocity is added in the flow direction to simulate a pressure gradient.
- The equilibrium distribution is calulated according to formula 14.34 of Thijssen 2007.
- The densites are relaxed at each point according to: $n_i^{new} = (1 - 1/\tau)n_i^{old} + n_i^{eq}/\tau$. In which τ is the relaxation time.

Usefull relations

The relation between the viscocity and the relaxation time is given by: $\nu = \frac{2\tau-1}{6}\frac{\Delta x^2}{\Delta t}$. (Thijssen 2007)

The curvature of the velocity of the flow should be $\nabla P/\rho v$. In which ∇P can be related to the velocity addition in the flow direction via: $\nabla P = c\rho\Delta v$. (Thijssen 2007)

The Reynolds number is calculated by: $\mathrm{Re} = \frac{uW}{\nu}$ in which u is the mean velocity in the pipe along the flow direction, W is the width of the pipe and ν is the viscosity of the fluid.

The simulation

The setup

For the geometry a 2D 'torus-pipe' is defined, in which a square or circular obstruction can be placed, which will be centered at $(0.5W, 0.25L)$ in the pipe. (It is called a 'torus-pipe' as the outflow is given as inflow to the 2D pipe.) Then the lattice vectors are defined, including a relationship between the lattice vector and its inverse (version in opposite direction). This is needed in a later stadium to incorporate the boundary conditions.

As for the parameters, Δx and Δt are set to 1 to simplify the relations. The maximum number of simulation iterations and the viscosity can be set to the desired value. The viscosity should be defined such that the relaxation time, τ, is bigger than 1. This is to ensure that the system does not relax faster than the minimal time step, Δt.

Application of the algorithm

The simulation is initialized such that the density everywhere equals 1 except for the boundaries and in the obstruction, at which it is set to 0. Then a velocity forcing in applied (due to the pressure gradient), after which the equilibrium distribution, n_{eq}, is determined for every lattice point.

After this, the algorithm described above is implemented. For the translation of the densities, `numpy.roll()` is used in combination with the lattice vectors.

The boundary condition has the following working principle: If the density corresponding to vector `[1,0]` crosses the right boundary, the density gets transferred to the `[-1,0]` vector, such that the next time step the density returns to the pipe domain.

The rest of the code is implemented straightforwardly.

The equilibrium stage

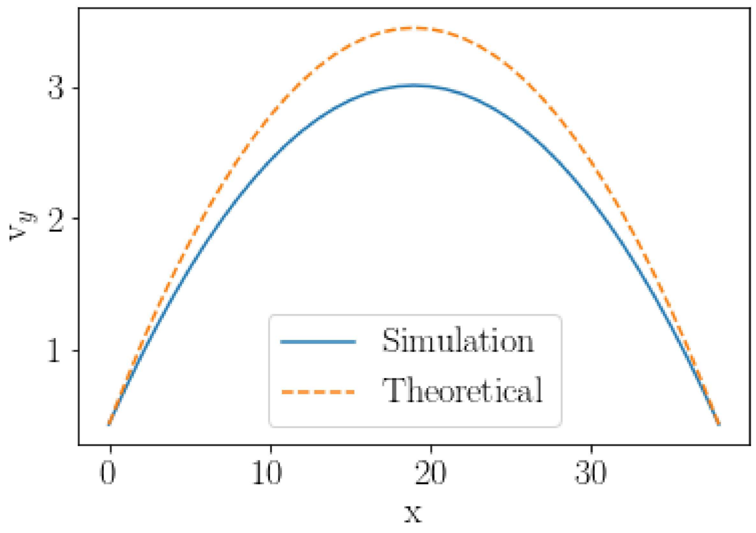
The simulation for the 2D 'torus-pipe' is stopped by the simulation once a global equilibrium has been reached. This equilibrium is based on the velocity in both x and y-direction in the geometry. Once the velocity in every grid point of the geometry does not change much with respect to the previous iteration $(v_i - v_{i-1} < \epsilon)$, the simulation is regarded to be in equilibrium and is therefore ended. ϵ is set to 0.001 which is about 2 orders of magnitude smaller than the flow velocity through the pipe (in the y-direction). The maximum amount of iterations is set to 1000.

Results and Discussion

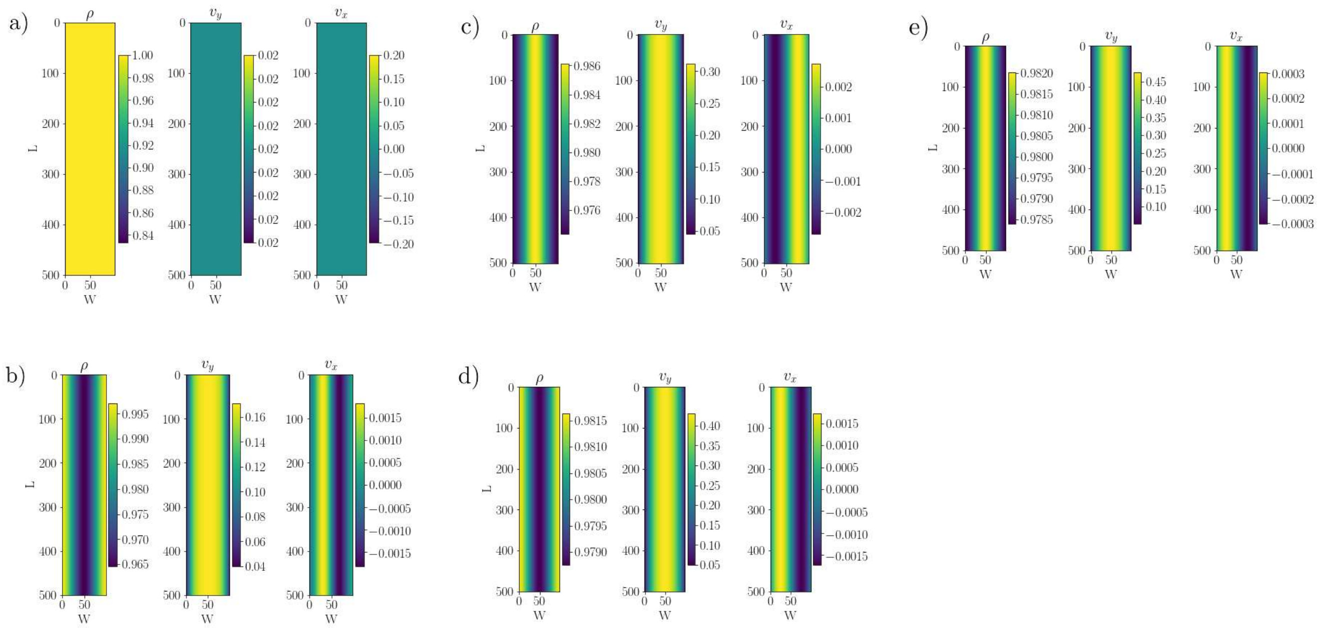
The results are shown for a 500 by 100 torus-pipe unless stated otherwise. Viscocity is set to 2 and the added velocity to 0.01. The diameter of the obstructions (in x-direction) is equal to 16. The Reynolds number in the pipe is between 14 and 17 for the results shown below. Animations have been made as well. The diameter of the obstructions in the animations is 32. (In order to make it better visuable.) The animations can be found in the **'images_and_animation' folder** and are bundled per situation.

Flow without obstruction

The flow in the 2D 'torus-pipe' up to equilibrium (for $\epsilon < 0.001$) is shown in the figure below. The equilibrium stage is reached after 344 iterations.



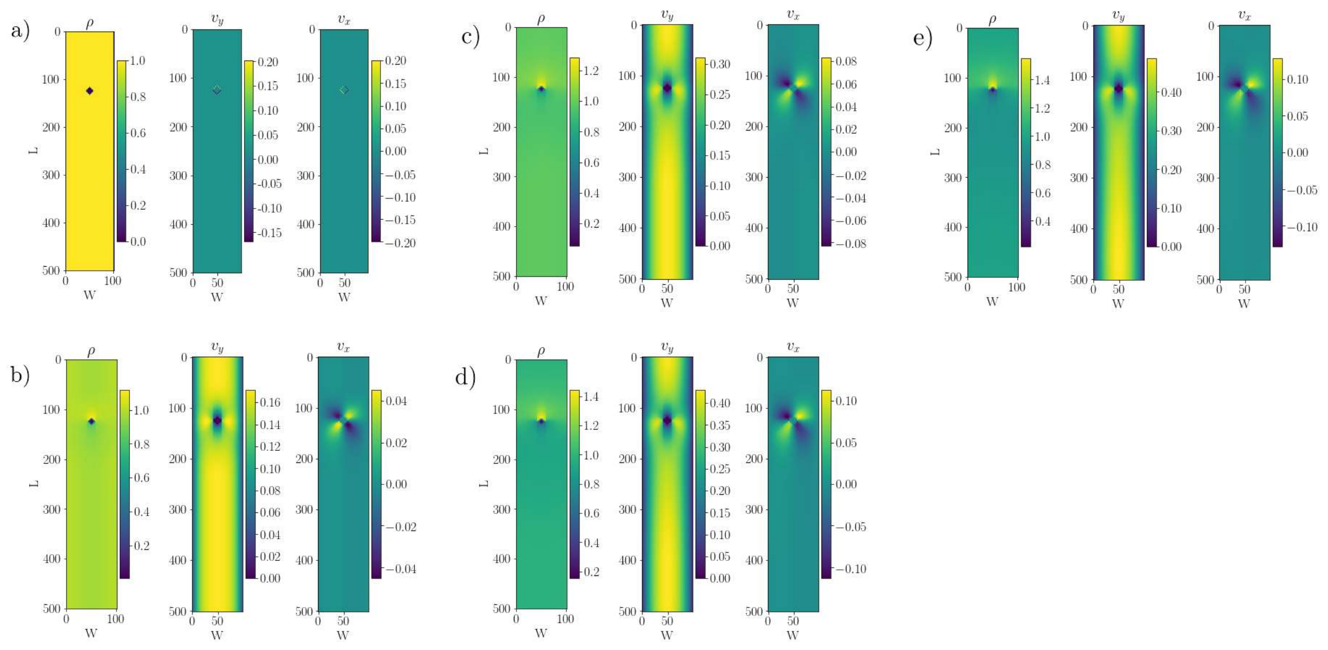
The contour plots incorperating, from left to right: density, velocity in the y and x-direction. It can be seen that the velocity in they y direction becomes a parabola as expected.



The subfigures a) until e) refer to iteration number 0, 100, 200, 300 and 344 respectively.

The plots above show that the flow goes to an equilibrium stage. Although the simulation is stopped after 344 iterations, because the velocity after 1 iteration is then below the errormargin, a weird oscillation in the velocity is seen. This is also visibile in the animations. As this is not seen for the case with an obstruction in the flow, we suspect that this arises form the initial, zero boundary conditions, creating standing waves in our plots. For the flow with obstruction these waves are interupted and thus don't occur.

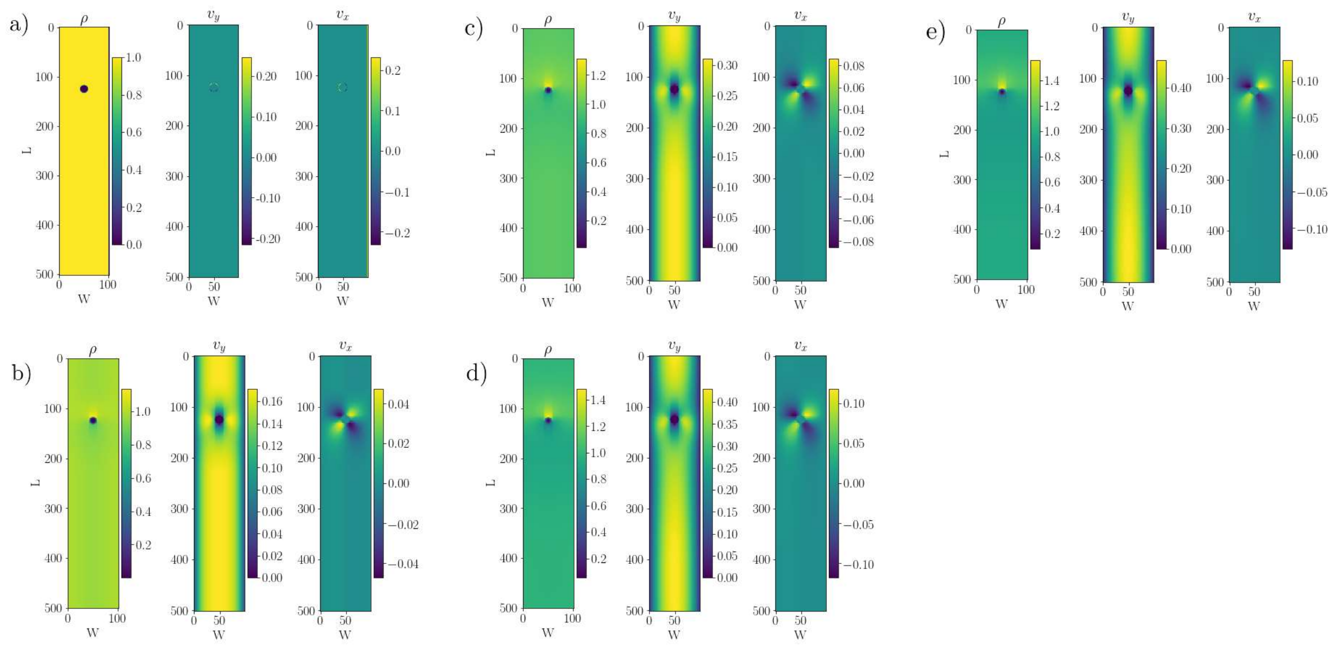
Flow around a square obstacle



The subfigures a) until e) refer to iteration number 0, 100, 200, 300 and 356 respectively.

The plots above show that an equilibrium is reached after 356 iterations. Nothing unexpected happens. Since the density is build up in front of the obstacle and lower after. Also the flow velocity around the object seems to be increased opposed to no object, which is in general what is expected.

Flow around a cylinder



The subfigures a) until e) refer to iteration number 0, 100, 200, 300 and 337 respectively.

The plots above show that an equilibrium is reached after 337 iterations. As the cylinder and the square have a similar size and the geometry are similar as well, the profiles look a lot alike. This is not very surprising.

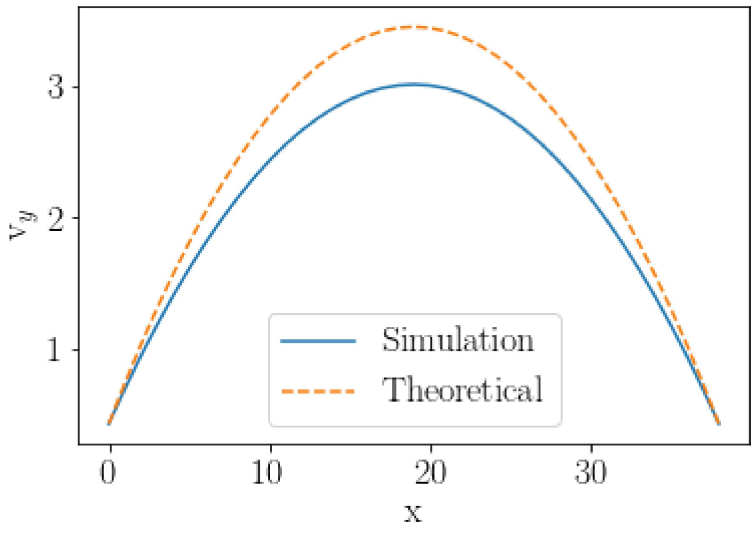
Conclusion

All in all, the code seems to work, but shows an unexpected oscilation in the velocity-profile of the flow in the 2D 'torus-pipe' without an obstruction in the flow. The origin of this behaviour is still not certain and requires further investigation. Some suggestion to improve the simulations are to implement different boundary conditions, in the flow direction, such that the outlflow does not equal the inflow of the pipe. Also the determinitation of the equilibrium state now happens by comparing the previous to the current state, it would be better if it is compared to a couple of states before the current state.

Code checks

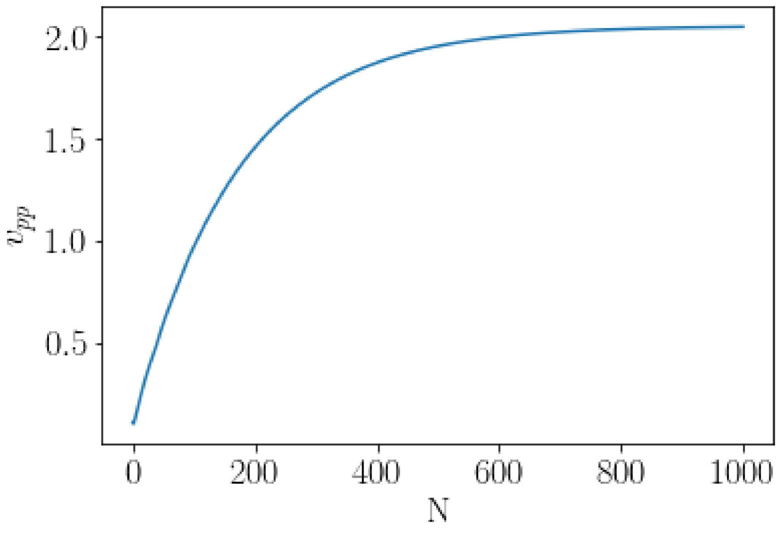
Parabolic flow profile

The flow profile seems to agree fairly well with theory. Altough for some settings (viscocity and velocity) differences are bigger. Unfortunately we don't have an explanation for this, other then that for some settings the simulated flow does not behave fully laminar. The figure shown below is for a system size of L=60, W=2.

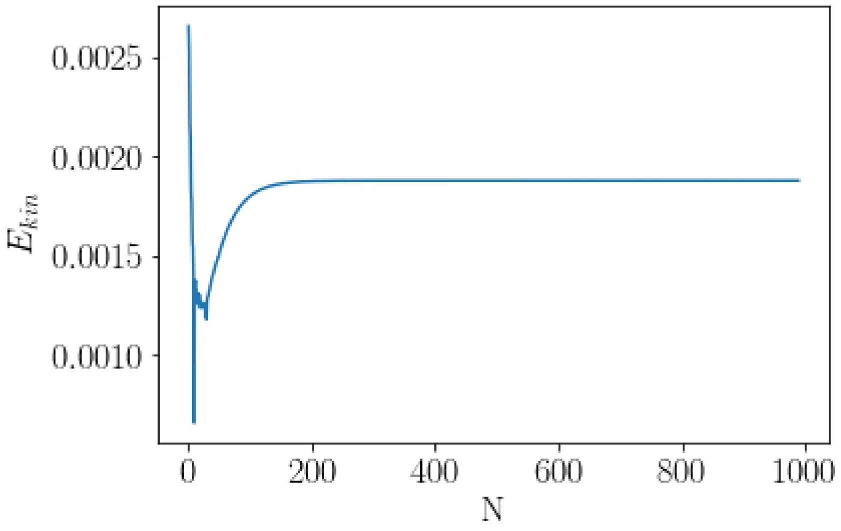


Conservation

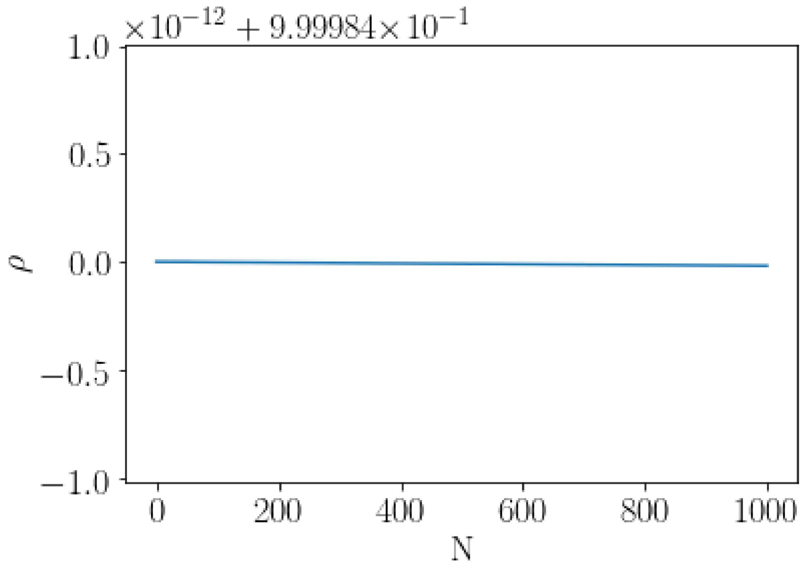
Equilibrium The velocity is initialized at zero, at every time step a small velocity is added which simulates a ΔP. At some point, the velocity should equilibrate. This behavior is indeed observed as can be seen in the figure below where the average velocity per particle is plotted vs the simulation iterations.



Energy: The kinetic energy in the system is shown in the plot below. It can be seen that after an initial period of added velocities in the flow direction, equilibrium is reached and kinetic energy is conserved. This equilibrium optically seems to be earlier reached than for the plot above due to the quadratic scaling of kinetic energy with velocity. Hence it should arrive at the same time in equilibrium as the speed plot. The start of the plot is due to the initial conditions.

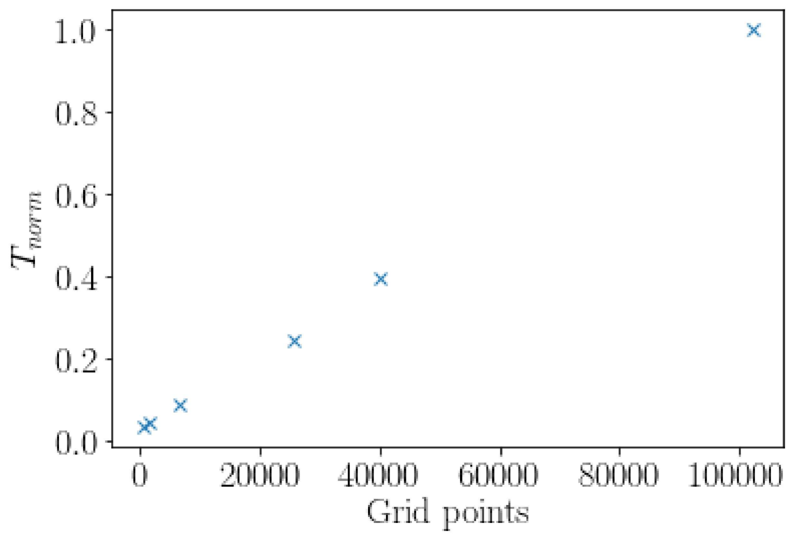


Density: In the following graph, it can be seen that the density averaged per particle is conserved during the simulation. Notice the scientific notation on the top. The average density is slightly lower than 1 since the borders are initialized on 0 density while the rest of the pipe is initialized at 1.



Simulation performance

As can be seen in the plot below, the simulation scaling is linear with the number of grid points (in the limit), hence quadratic with sytem size. Overal for 1000 iterations and 100,000 grid points one should expect a simulation time of around 1.5 minutes.



Collaboration

This time collaboration proved to be a bit harder due to increased load from other courses. This meant that Vincent had a bit more time at the start, while Tom in the later parts of the project. We solved this by letting one be more concerned with the setup of the code and the other by checking and optimizing the code. The project itself was, compared to the others, more difficult for us, due to several bugs we encountered (for example the weights in the equilibrium distribution formula as discussed in the text). In the end, we managed to solve most of them, although some behavior is still unexplained. For the next project, a takeaway is to be both more involved at the start. Such to set out the code structure together since this makes errors less likely to happen and leads to more logical and readable code.

Both: Boltzmann Lattice algorithm, data-processing, report, overal small tweaks

Tom: Animation, alogrithm optimisation

Vincent: Initial algorithm implementation

Sources

- Thijssen, J. (2007). Computational physics, 2nd edition. Camebridge University Press
- Bao and Meskas (2011). Lattice Boltzmann Method for Fluid Simulations. url: https://cims.nyu.edu/~billbao/report930.pdf (https://cims.nyu.edu/~billbao/report930.pdf) Retrieved on 31-05-2018