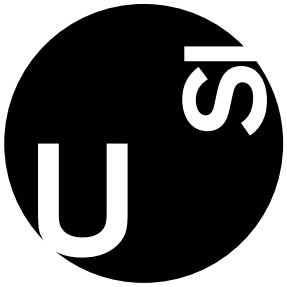

Università
della
Svizzera
italiana

Faculty
of
Informatics

Bachelor Thesis

June 14, 2023

Implementing Fluid Dynamic Simulation using DPD Method

Francesco Costa

Abstract

Digital simulations are useful tools to visualize results, extract information and simulate physical phenomena. There are many methods that can be used to create simulations and each have their own advantages and flaws. One main characteristic to differentiate these types is the length-scale: atomistic models can be used to represent materials and physical objects at the atomic level, leading to precise but computationally expensive simulations; macroscopic models are focused instead on bigger systems, with the advantage of being less computationally demanding. DPD (Dissipative Particle Dynamics) is a method that stands in between the aforementioned scales. By considering groups of atoms as particles we can simulate a fluid in a real time simulation (which would usually be done only in static environments with fixed objects). The aim of this project is to extend an existing physics simulation tool by implementing the DPD method. Simulations of this type are used in a number of different fields, such as the medical one (simulating blood flow in vessels). In our case the purpose of this tool is to mainly be used for didactic means, to show the power and possibilities of informatics.

Advisor
Prof. Antonio Carzaniga

Advisor's approval (Prof. Antonio Carzaniga): Date:

Contents

1 Introduction

Simulating consists in the process of imitating a system or real-world phenomena. This can be done in many different ways but requires in general one or multiple models that represent the behaviour of the target in the simulation. The choice of such model and its definition are key to create an environment which provides valid and reliable results.

There are a number of fields that make extensive use of simulations for different reasons. In some cases, such as for engineering, the system of interest may be dangerous to test, or it may still not have been built. Using simulations allows to test a system before it goes in production, decreasing time and cost in the development stage by applying the trial and error method in a controlled and flexible environment. There is in general an interest in modeling natural systems (a physic system in the case of this project), as well as human systems (commonly economics). This need brought forward an increasing necessity of bigger and more complex simulations, leading us to the role of informatics and computer simulations.

1.1 Computer Simulations

The power of informatics in the context of simulating is the prime example of why it is important for workers and researchers in different fields to learn the basics of computer science and programming. Depending on the complexity and suitability of the model it is possible to simulate virtually any system (accepting in some cases higher degrees of approximation). Another advantage of digital simulations is the possibility of visualizing the results using computer graphics techniques.

Most simulations of natural systems are based on physical laws, thus reducing the problem to the definition of a physics engine. While in general simple physical laws can be reproduced in software, there are cases in which it is only possible (or advised) to use approximations and simplifications in order obtain better performance. The cost of the simulation and the computational power available act as a limit in the way certain phenomena can be modeled. Real-time simulations are based on models that can execute at the same time as the real-world system, allowing to observe very realistic simulations. The problem with this approach is the computational strain that is forced onto the machine running the program, which means that for certain complex simulations either a really powerful machine or a very efficient model and code are required.

1.2 Approaches

Simulations can vary in kind vastly depending on the system they are based on. One big distinction can be made when talking about imitating natural systems and that is length-scale. A simulation of a chemical compund to the atomic level will be very different from one representing the effect of gravity on rigid bodies. This difference can be categorized in three main classes:

Atomistic scale Simulations of this length-scale aim to study the composition of matter itself, resulting in usually computationally expensive processes. The main fields implementing such models are biology, chemistry and physics, especially when the three disciplines intertwine.

Macroscopic scale When describing the global state of a system we don't focus on the single elements but rather on generalizing the dynamics of the phenomena. The model usually consists of sets of equation describing the system (such as equations of motion for Mechanics and Kinematics).

Middleground between scales There are cases in which we would want to describe matter in fine detail without incurring in excessive execution cost. One solution to this problem is to consider particles instead of atoms. These particles are agglomerates of atoms which allow for reasonably precise control of interaction between the elements of the system as well as a good overall representation of the global state. Using a method built on such a length-scale allows for example to simulate hydrodynamic behaviour at a much lower cost.

Fluid Dynamics being a focal point of this project it is important to talk about the difference between CFD (computational fluid dynamics) simulations and real-time simulations. In CFD the objective is to imitate the behaviour of a fluid usually around static objects. This is done to study the characteristics of both elements in the simulation. If we want to consider instead the fluid as an environmental effect, we would want to be able to simulate foreign bodies moving in such fluid. This is not possible with the standard CFD approach since the cost of emulating the fluid is too high to be incorporated in a real-time simulation. The method used for this project allows for such a simulation without compromising excessively on the behaviour of the fluid.

1.3 Goal of the project

The goal of this project was to extend an existing tool consisting of a simple physics engine capable of modeling the effect of gravity on 2D spheres and the collision between those bodies. The choice of direction for the extension was influenced by my interest in the topic of fluid dynamics and by being introduced to the method used by Professor Pivkin. The purpose of the tool also played a major role in the design process, since as a didactic tool to showcase the possibilities of computer programs it was most important to aim for a visually interesting and pleasing result. This fact didn't diminish nontheless the interest in also applying and validating the particular method used for modeling.

2 Fluid Dynamics Simulations

Fluid mechanics is a branch of physics concerned with the behaviour of fluids (such as liquids or gases) when forces are applied to them. Given our objective and the nature of simulations we will mainly focus on the dynamics of fluids. The study of the motion of a fluid involves the correlation of many of its properties such as velocity, density, pressure and temperature. The results of such models are used in a number of different disciplines like (most commonly) aerodynamics and hydrodynamics.

2.1 CFD Simulations

The most common way to model a fluid is through Computational Fluid Dynamics (or CFD) simulations, which, as introduced in 1.2, allow to reproduce the bahaviour of a fluid in an extremely detailed way. The cost of modeling systems using this method is usually expensive, hence making CFD not suitable for real-time applications such as the one developed in this project. The theory and application of this method still makes it worthwhile to study since it gives better insight on why other approaches may or may not work and why they may work better in certain scenarios.

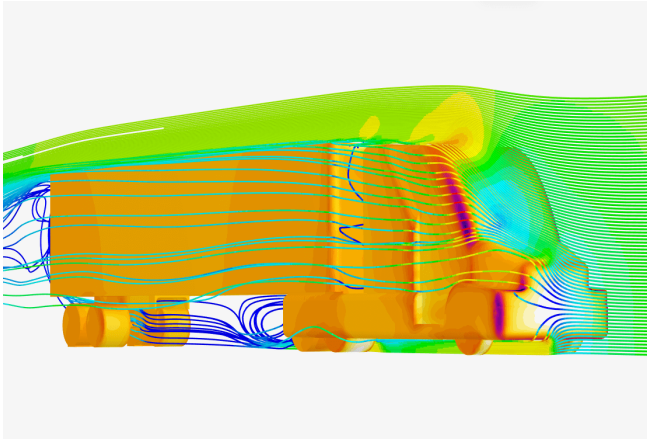

Figure 1. CFD simulation of fluid (Taken from SimScale)

As the name would suggest CFD is a technique based on mathematical computation and numerical analysis, which explains why powerful computers are necessary to create and handle the simulations. The complexity of the system to be modeled is directly correlated to the computing power required to simulate it.

In the following section a theoretical backgorund on the implementation of this method is given in order to accustom the reader to the crucial recurring elements of fluid dynamics.

2.1.1 Navier-Stokes Equations

The Navier-Stokes equations are differential equations used to describe the behaviour of a fluid at a specific point and time. The solution of the equations is effectively a vector field for the velocity of the fluid in each point of interest. Given the velocity field the equations can also relate the other important elements characterizing fluids, namely

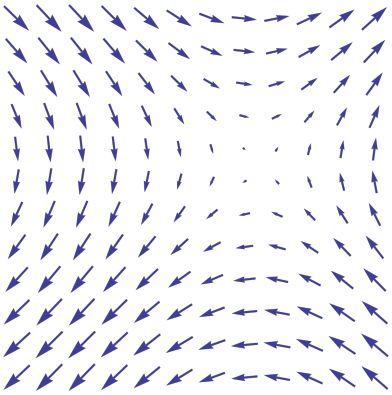

Figure 2. Vector Field to represent for example velocity in each point (Taken from Wikipedia)

pressure, temperature and density. The Navier-Stokes equations for an incompressible fluid with velocity field $\vec{u}$ are

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u}$$

where

- t is the time;
- ∇ is the divergence (which operates on a vector field to give a scalar field);
- ρ is the density;
- p is the pressure;
- ν is the kinematic viscosity (which correlates the dynamic viscosity to the density of the fluid).

Depending on which fluid we are trying to model the Navier-Stokes equations change following the properties of the fluid, but for most cases they remain eqautions for which the solution can only be found with the help of computers.

2.2 DPD Method

Dissipative particle dynamics (or DPD) is a simulation method first introduced by Hoogerbrugge and Koelman [1] used to model hydrodynamic behaviour. The implementation process applied in this project takes inspiration from a paper by Robert D. Groot and Patrick B. Warren [2] which uses the formulation of DPD by Español and Warren as a starting point [3]. The premise of the method is to consider particles as agglomerates of atoms that interact with each other. This formulation leads to a simulation at a length-scale larger than the atomistic scale, but smaller than the macroscopic scale. This intermediate length-scale allows the method to model the properties of the fluid with sufficient detail while avoiding the cost of simulating each atom individually.

The usual set of equations (Navier-Stokes) used to model the behaviour of the fluid is hence replaced by a new formulation of the forces between particles. These forces must have specific definitions in order to satisfy the conservation of energy and momentum, and they consist of

Conservative Force This is a simple repulsion force that acts on the line between centres of the particles. It is to be calculated for each pair of particles in the system.

$$\mathbf{F}_{ij}^{C} = \begin{cases} a_{ij}(1 - r_{ij})\hat{\mathbf{r}}_{ij} & (r_{ij} < r_c) \\ 0 & (r_{ij} \geq 1) \end{cases}$$

where a_{ij} is the maximum repulsion between particles i and j, and r_c is the unit of lenght for our system. Since we represent the particles as circles of a certain radius, the unit length will be set to the diameter of the particles. Finally we define distances between particles as

$$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j, \quad r_{ij} = |\mathbf{r}_{ij}|, \quad \hat{\mathbf{r}}_{ij} = \frac{\mathbf{r}_{ij}}{r_{ij}}$$

Dissipative Force This force simulates the friction between particles and removes energy from the system.

$$\mathbf{F}_{ij}^{D} = -\gamma \omega^{D}(r_{ij})(\hat{\mathbf{r}}_{ij} \cdot \mathbf{v}_{ij})\hat{\mathbf{r}}_{ij}$$

where $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ is the difference in velocity between particles; γ is in relation with the amplitude σ of the random force and defined as $\sigma^2 = 2\gamma k_b T$ (which will be discussed in later sections); ω^D is a weight function dependent on the position of the particles and defined as

$$\omega^{D}(r) = \begin{cases} (1 - r)^2 & (r < 1) \\ 0 & (r \geq 1) \end{cases}$$

Random Force This force is used to balance the system and avoid a situation of stall that would occur if only the first two forces were present.

$$\mathbf{F}_{ij}^{R} = \sigma \omega^{R}(r_{ij})\theta_{ij}\hat{\mathbf{r}}_{ij}$$

where σ is the noise level which has been set to 1 for our purposes; θ_{ij} is a random variable unique for each pair of particles; ω^R is a weight function related to ω^D with the form

$$\omega^{R}(r) = \sqrt{\omega^{D}(r)}$$

Now that we have defined each force individually, we can update the resulting force on each particles by summing them together

$$\mathbf{f}_i = \sum_{j \neq i} (\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R) \tag{1}$$

2.2.1 Eqution of motion and parameters

In order to update position, velocity and forces acting on the particles a system of equations of motion is necessary. Since we are dealing with particles in the case of molecular dynamics, a modified version of the velocity-Verlet algorithm (an integration method for equations of motion in cases such as ours) is used in this case, resulting in the set of equations

$$\begin{cases} \mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + 1/2(\Delta t)^2 \mathbf{f}_i(t) \\ \tilde{\mathbf{v}}_i(t + \Delta t) = \mathbf{v}_i(t) + \lambda \Delta t \mathbf{f}_i(t) \\ \mathbf{f}_i(t + \Delta t) = \mathbf{f}_i(\mathbf{r}(t + \Delta t), \tilde{\mathbf{v}}(t + \Delta t)) \\ \mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + 1/2 \Delta t (\mathbf{f}_i(t) + \mathbf{f}_i(t + \Delta t)) \end{cases}$$

Since the force depends on the velocity of the particles the idea is to make a prediction for the velocity, namely $\tilde{\mathbf{v}}$y which is then used to calculate the force and finally the new velocity. The variable factor λ accounts for some interactions in the system but for our purposes we will fix it to $\frac{1}{2}$ as done in the experiment we take inspiration from.

Given that we are not trying to study the effects of the different parameters and the stability of the model, we will generally fix most variables to values known to produce stable simulations. Since the research by Groot and Warren is instead focused on studying how the simulation is affected by the choice of parameters we will generally take inspiration from their results to implement our model.

The remaining variables that need to be defined are

Repulsion parameter a The variable is used to describe the maximum possible repulsion between particles. The higher this parameter the more entropy can be observed in the system. For this project the value was set to $a = 35$, which resulted in a satisfying simulation.

Temperature of the system T The temperature of the fluid is one of the main properties that can be studied, since it has strong consequences on the stability of the simulation. For our purposes we will work on the assumption that $k_b T = 1$, which simplifies the problem and allows for the direct comparison of noise parameters $\frac{\sigma^2}{2} = \gamma$.

3 Implementation

As previously mentioned this project is based on an existing simple physics engine developed by Professor Carzaniga. The simulation program was originally written in `C` language and later translated in `C++` in order to access powerful methods and data structures. The objective was to build on the provided tool by adding new interesting features while keeping the essence of the simulation. This specific extension required a number of additional concepts and ideas but uses the base basic objects and processes as the original.

3.1 Existing tool and technical background

3.1.1 Life cycle of the program

The original program implements a state based simulation where, after initializing the starting state, the system is updated at each frame (defined by the decided time step). The operations executed at each time step define the behaviour of the simulation and are the core of the implementation. Outside of updating the state of the system, the program also manages the termination of the process.

3.1.2 Cairo graphics library

The only external library used for this project is *Cairo*[1], which allows us to visually represent the objects of the system and the simulation. It is a 2D graphics library supporting multiple output devices. The cairo API implements operations similar to those of PDF and PostScript, such as stroking. All drawing operations can also be transformed through affine transformations (such as rotation and scale). The library is originally written in `C` language but provides bindings for several programming languages.

Cairo relies on a three-layer for the drawing process[2]. The steps can be divided in

- creating a mask which consists of basic vector primitives or forms such as circles, squares or even Bézier curves;
- defining a source which may be a color, a gradient or vector graphics, and with the help of the previously defined mask a die cut of the source is made;
- finally the result is transferred to a destination surface which will be provided by the back-end to the output.

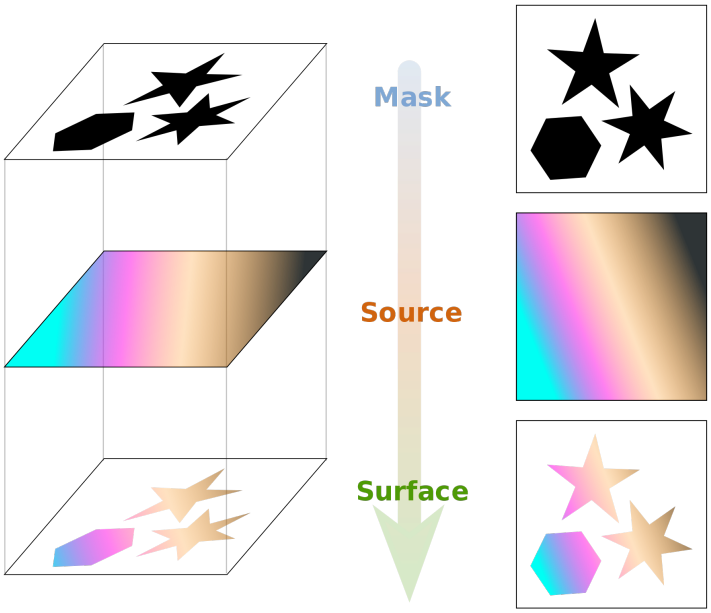

Figure 3. Cairo drawing model (taken from Wikipedia)

[1] https://www.cairographics.org/documentation/
[2] https://en.wikipedia.org/wiki/Cairo_(graphics)

3.2 Implementing the DPD method

The actual implementation of our method required a number of different steps that range from definition of the behaviour of the particles to the reimagination of the space in which the balls move. The latter improvement was introduced with the objective of making the simulation more interesting and appealing visually, while also testing the model under different constraints.

3.2.1 Defining motion of particles

The first part of the implementation process revolved around taking the theoretical results shown in the pepers studied and put them into pratice in the code. The existing structure was a good starting point since it already updates the state of the objects through equations of motion at each time step. In order to introduce the DPD method, the starting point was to redefine such equations (using the velocity-Verlet algorithm 2.2.1) and to implement the function used to update the forces on the particles (as per equation 1). One of the few architecture details that needed to be changed was the possibility of calculating the position of the particles separately from the other updated properties. This has to be implemented to satisfy the new equations of motion introduced.

Another extension was the introduction of various extra fields for the ball class, which spares us the need of creating a new construct for the particle objects. This new information allows us to discern various differences between fuild particles and those defining the boundaries. On the same vein a number of control variables were defined to ease the ability of controlling the simulation when testing. Finally a new control parameter was added to modify the velocity of the flow dynamically during the simulation.

3.2.2 Defining new objects

The main addition that allows to define flows of motion for our simulated fluids are the border particles. This is one of the proposed methods of containing the particles of the system, and consists in assigning a number of objects to move around the edge of the area of interest with the objective of inducing movement in the system and constraining the particles inside. For the purpose of the project, simple elastic collision were also used to keep the particles in check, reducing the importancy of the border elements to simple flow controllers.

With the main interactions defined, the focus went back to one of the most important requirements of the simulation: to be interesting and visually appealing. The simple vertical flow was enough to test the behaviour of the system, but not particularly effective in showcasing the power the model provides. The next improvement was to implement a track the fluid could follow, introducing more complex movement and possibilites for the particles to interact in different scenarios. The construction of such a path required the definition of a new structure. The final choice for the shape of the track came down to a general polygon, since they allow high flexibility and don't introduce excessive complexity in the system. A polygon is simply defined by its vertices and a path can be defined by creating two similar polygons with different sizes, representing inside and outside borders. After creating the inside polygon we can generate the corresponding outside polygon in many different ways. In our case this was done by simply extending the vertices following the direction of the line splitting the corners of the polygon in half.

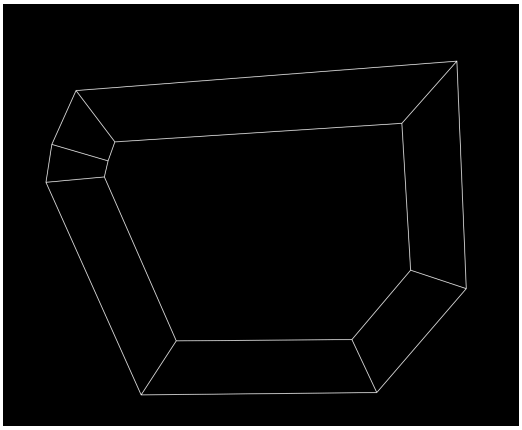

Figure 4. Example of a track generated randomly

This simple procedure allows us to generate the track randomly and, with sufficient constraints, we could minimize the possible problems arising from such a direct method. Having both polygon well defined by their vertices it was then possible to easily check for collisions of particles on the boundaries using simple algebra.

One recurring problem was the choice of spawn for each particle at the start of the simulation. The wish to randomize the generation of the polygon in order to get different and interesting simulations, forced the starting state of the particles to also be generalized for each possible situation. The main issue is that the length of each segment composing the polygon is highly variables depending on the position of the vertices. The solution to the problem of generating the particles was to give a different weight to each segment, depending on their length, and create an according number of balls along each of them. This method reduces drastically the initial clustering of particles, minimizing possible problematic starting states.

3.2.3 Auxiliary functions

Given the nature of the simulation, 2D vectors were used for most of the implementation, leading to the need of the `vec2d.h` file which compiles, in addition to the data structure itself, a number of methods useful for our purposes. The header file was already present, but it was extended to include operations such as the cross product and the module of 2 dimensional vectors.

In order to define an order for the vectors of the polygons, a comparator function was also defined, with the objective of sorting points in the space in clock-wise order. This is needed for cairo to correctly draw the polygon and to define the segments.

4 Optimizing the Simulation

As previously mentioned the topic of computational cost is important in the context of this project because fluid-dynamics simulations are usually extremely expensive to run. In order to preserve the nature of the original tool, the DPD method was chosen to emulate the behaviour of fluids.

4.1 Computational cost

Usual CFD simulations on an atomistic scale involve high computational cost given by the sheer number of elements that need to be simulated. Using the proposed method allows to cut significantly this cost and introduces the possibility of getting meaningful results by optimizing the calculations.

The method simulates the interactions between particles by calculating resulting forces for each individual object. Since each particle can in theory affect all the others in the system, the complexity of our method results in $O(N^2)$, for N total particles. The exponential nature of the complexity means that the process quickly becomes very expensive for an increasing number of elements added to the simulation. One of the main issues with the formulation of our program is that particles affect each other only if they are within a certain cut-off distance, but with no prior information we still have to check all possible interactions. The solution to the problem is to keep track of the position of each element in order to gain fast acces to the neighbourhood of the particle considered during the calculations. One such simple but effective structure is the cell list.

4.2 Cell list

A cell list is a data structure commonly used in molecular dynamics and consists of a simple grid. By dividing the simulation domain in cells of length greater than or equal to the cut-off distance of interaction between atoms (or particles in our case), we can easily extract how big a neighbourhood we have to consider in order to include all relevant elements in the calculations. The total cost of finding all pairwise distances between particles becomes thus effectively linear, with complexity $O(N)$ on the number of elements.

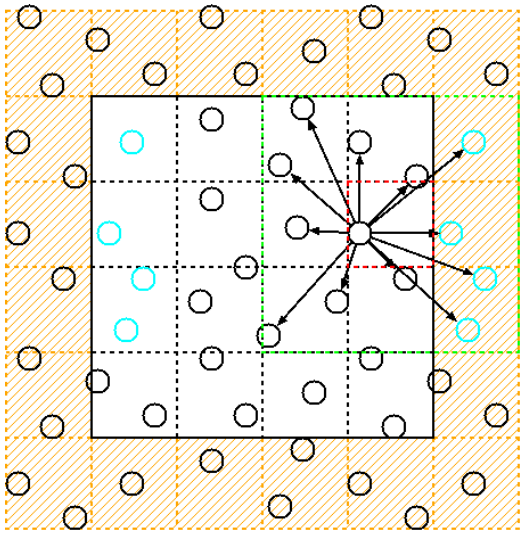

Figure 5. Cell list with ghost cells boundary condition [3]

One implementation detail that requires particular attention are boundary conditions. The most common approach is to use periodic boundary conditions, which are preferred to artificially imposed constraints. One such method is the ghost cells approach shown in figure 5. By adding an external layer of cells we can represent the periodic nature of the system, saving us the need of specifying a particular behaviour for the cells lying at the edges of the simulation domain.

4.2.1 Observed improvement

A simple expertimental test was run to asses the effectiveness of cell list in reducing computational cost of the simulation. The test was run with a varying number of particles while trying to keep all possible confounding factors and variables constant.

[3] By Pedro Gonnet - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=117923800

N Particles	No optimization	Cell list optimization
140	48.2	45.0
200	54.5	48.0
400	76.4	54.0
600	98.5	61.4
1000	100 (frozen)	73.4
2000	100 (frozen)	91.6

Table 1. Average CPU Load (%) comparison depending on number of particles

As we can clearly see from the experimental results, the improvement provided by the cell list increases more than linearly with respect to the introduction of particles in the system. This is due to the difference in complexity when optimizing the calculations comparatively to going through each particles. The switch from quadratic to linear complexity can be noticed in the observed results.

5 Conclusion

6 Appendix

References

[1] P. J. Hoogerbrugge and J. M. V. A. Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *Europhysics Letters*, 19(3):155, jun 1992.

[2] Rob Groot and Patrick Warren. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. *The Journal of Chemical Physics*, 107, 09 1997.

[3] Pep Español and P.B. Warren. Statistical mechanics of dissipative particle dynamics. *EPL (Europhysics Letters)*, 30:191, 05 1995.