CONTACT PROPERTIES OF HYDRATED SILICA SURFACES

by

André Douzette

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

January 2016

Abstract

In this thesis a molecular simulation program is constructed from scratch. It utilizes the Vashishta-Kalia-Rino-Ebbsjø many-body potential model in an efficient parallel implementation. This implementation is validated against physical known properties of silica and water which are known from experimental data. The implementation is then used to investigate a contact problem of a cylinder and half space, both consisting of silica. Effects of surface hydration, and surface roughness are investigated. The results suggest that hydrating silica surfaces will result in lower pressure in the area of contact. In addition the reduced elasticity modulus of a rough surface has a slightly smaller value compared to a smooth surface, according to data gathered from the performed simulations.

Acknowledgements

When I now write the last paragraphs of my master's thesis, the years I've spent being a student come to a close.

I would like to thank my two supervisors for making this thesis possible. Professor Anders Malthe-Sørenssen, for his guidance. And Anders Hafreager for his support, motivational talks and excellent guidance. Without his help, this thesis would never have become a reality.

Contents

Chapter 1

Introduction

Contact mechanics was introduced by Hertz in 1882, as a continuum mechanical theory. In recent years the theory has been applied to smaller and smaller scales. It has long been known that classical models can not be used in the scale of nanometers. At this scale, atomic structures and fluctuations will play a big role in the resulting dynamics of the system. Investigations into contact of surfaces which approach the atomic limit are therefore a relevant research field.

Atomic fluctuations and roughness are in the continuum theory assumed to play no role in the resulting contact properties. However, lately it was shown by Robbins [1] that the atomic structure of a surface will greatly affect the properties of the contact behaviour in the atomic limit. His experiment utilized an ideal system which consisted of particles interacting with the Lennard-Jones potential [2]. Investigating this type of experiment by the use of a more advanced particle interaction model, one will be able to give insight into whether more realistic systems will behave similar to the dynamics described by Robbins results.

Contact of macroscopic sized rough surfaces may also be affected by the nanoscale structures. This is due to rough surfaces in contact which will have a contact area composed of many very small and discrete areas. This has been shown by Klafter et. al. [3].

Investigations into Robbins hypothesis would ideally be performed using quantum mechanical simulations. Quantum mechanical simulations usually scale by a high ordered monomial which is a function of the number of particles in the system. Even though computing power is still increasing with the rate predicted by Moore's law, the computing capacity is not at a point where performing such computations on large systems are possible. A simplification in order to reduce the computation required is therefore needed. One such simplification of quantum mechanic is molecular dynamics, where atomic interactions are modeled by classical potentials. It is a N-body type of simulation, where particle trajectories are calculated using potentials developed from quantum mechanical calculations. These types of simulations result in data of the micropscopic nature of the system, such as particle trajectories and velocities. Using this data, macroscopic properties of the system may be found. These macroscopic properties include pressure of two bodies in contact, or friction forces.

1.1 Goals

The purpose of the work performed in order to write this thesis, has been to create a working molecular dynamics code from scratch. The code will then be evaluated against known physical quantities, before an investigation of the contact properties of silica will be performed. The main goals of the thesis can be summarized as follows:

1. Develop a molecular dynamics implementation of a silica-water system using the clay force field model.
2. Develop a parallelized implementation of the model. The code will be optimized for the potentials which are studied.
3. Implement the reactive USC force field model.
4. Implement the Nosé-Hoover thermostat in order to study stiff systems in the canonical ensemble.
5. Make estimations about the contact properties of different structures of silica, and investigate whether the presence of water plays any role.

1.2 My contributions

This thesis is focused in the field of computational physics. The majority of the work done has therefore been computer science related, in the form of code development. A total of approximately 18000 lines of code has been written. Both force field models which are studied in this thesis, have been implemented by a C++ program written from scratch. The interface of this program is written in Python, which is able to handle the processes of initializing an atomic system, compiling and starting the C++ simulation program, and analysing the resulting data.

A lot of effort has been put into making the code as efficient as possible. Therefore the C++ program is written as an efficient implementation using open-MPI, a parallel message parsing interface, in order to parallelize the code. There is, however, more room for improving the performance of the implementation, which is discussed in chapter 8.

Writing the implementation from scratch is a benefit, but may also be a disadvantage to some degree. The benefit is that one will get a lot of insight into the models and procedures used to perform the integration of particle motion, and the ones used to speed up the calculation processes. Also when implementing new additions, additional research into the inner workings of the software is not required, as one already know every detail there is to know about the program. The disadvantage is the fact that writing code takes a lot of time. Often things do not work as intended, and several days at worst may be spent in order to fix the problem. This results in less time to spend on performing simulations of physical systems. One would perhaps be able to have a more in depth study of the physics of a problem, if already developed software was used.

In order to perform simulations of the canonical ensemble, a method of evaluating the coefficients of Noseé-Hoover chain, using Crank-Nicolson discretizations, were developed. This is described in section 4.3.

Contact experiments and simulations usually assumes the surfaces in contact are dry. This is an assumption which may be fulfilled on the macroscopic scale. However, on the atomic scale, hydrophile surfaces will always attract some amount of water. This is due to the fact that the Earth's atmosphere contains water, and removing all water from an experimental setup is very hard, if not impossible to achieve. We will therefore perform an experiment in which we investigate the effects hydration have on the contact properties of silica. Investigations into whether the surface structure plays any role are also performed. This experiment is described in section 7.3.

The program developed as a part of this thesis, may be found here at this adress: https://github.com/AndreDouzette/MolecularDynamics

Chapter 2

The molecular dynamics model

Molecular dynamics is a numerical method which is capable of calculating the dynamics of large systems of atoms. The basis of a molecular dynamics model is to approximate the complex quantum mechanical behaviour by a classical model. This means that particles are modelled as point particles, and their trajectories are described by their position $\mathbf{r}$, velocity $\mathbf{v}$ and the forces acting on them $\mathbf{F}$. In this chapter we will assume that the forces between particles will be described by short range and currently unspecified potential. A short range potential is a potential which is effectively zero at distances larger than a cutoff distance r_{cut}. By this approximation, it is possible to introduce different methods which speeds up the calculation of interatomic forces by possibly orders of magnitude. The shape of the potential is not important when introducing the force calculation methods of this chapter. However, for a list of examples of such short range potentials, the reader is referred to chapter 5 where the potentials used to perform the experiments mentioned in this thesis, are described in more detail.

After introducing the force calculation methods, we will look at how to parallelize the code in order to take advantage of the maximum computational resources available in a computer. This also enables the code to be run efficiently on a high performance computing cluster, such as Abel at UiO. Lastly we look at how the numerical time integration procedure is carried out, which enables the program to use Newton's equations to evolve the system in time.

2.1 Integration procedure

In order to perform a simulation of a molecular system, we need an initial state. This state is either created prior to the simulation, or it is loaded from a saved file from a previous simulation. This saved file should be a snapshot of a systems state, and therefore needs to include all particle positions and velocities. If the system is created, some care should be taken to make sure the initialized system is as close to the system we want to study as possible. For how silica and bulk water systems are initialized, see chapter 6.

After the system is initialized, the integration procedure is started. This procedure is composed by the force calculation, the time integration scheme, parallel communications, and if desired, statistics sampling of the system. The barebones of the simulation procedure is sketched out in Algorithm 1.

Algorithm 1 The flow of a molecular dynamics program

```
1: Read or initialize an atomic structure.
2: for $t \in time$ do
3:     Calculate Forces
4:     Integrate particle positions
5:     Communicate particles between processors
6:     Save statistics
7: end for
```

The step which is by far the most computationally expensive is the force calculation. Every particle will exert a force onto every other particle in the system. Therefore the number of particle interactions in the system is $N(N-1)$, where N is the number of particles. By implementing a straightforward naive approach, the time it takes to calculate interatomic forces will scale quadraticly with the number of particles. It is therefore crucial to optimize this step in order to be able to simulate large systems.

2.2 Linked cell lists

Reducing the $\mathcal{O}(N^2)$ scaling is a crucial part of any molecular dynamics program, as this enables us to perform computations on larger systems, consisting of up to houndreds of thousands of atoms. An easy way to reduce the quadratic scaling to a linear one, is through so-called linked cell lists. When assuming a cutoff r_c of short-range potentials, and there are no long-range potentials to evaluate, particles will only interact with other particles within the cutoff range. Particles within cutoff range of another particle will be gathered in the particle's neighbour list, in order to allow forces between these particles to be calculated later.

If the simulation space is much larger than the cutoff length, it will be beneficial to sort particles into cells depending on their positions. The simulation space is therefore partitioned into multiple boxes whose edge lengths are equal or slightly larger than r_c. With a simulation size S_i, where i is the dimension x, y or z, the total number of cells is $n = n_x n_y n_z$ and $n_i = \lfloor S_i/r_c \rfloor$. The cells' edge length will be

$$L_d = \frac{S_i}{n_i} = \frac{S_i}{\lfloor S_i/r_c \rfloor} \geq r_c \tag{2.1}$$

The particles located in each box are added to a linked list. Particles will only interact with other particles in their own cell and the surrounding 26 cells, as all other particles will be out of range.(see figure 2.1 for a 2D graphical representation).

By using $V_c = L_x L_y L_z$ as the volume of the cells and $V = S_x S_y S_z$ as the simulation volume, the number of distances between particle pairs which are calculated, will be linearly scaled with the number of particles in the system as $27N\rho_p V_c$. Here ρ_p is the particle density, which is assumed to be constant when scaling. This is a reasonable assumption, as one would usually look at a system with the same properties when scaling, and the average particle density of the system would therefore be kept constant.

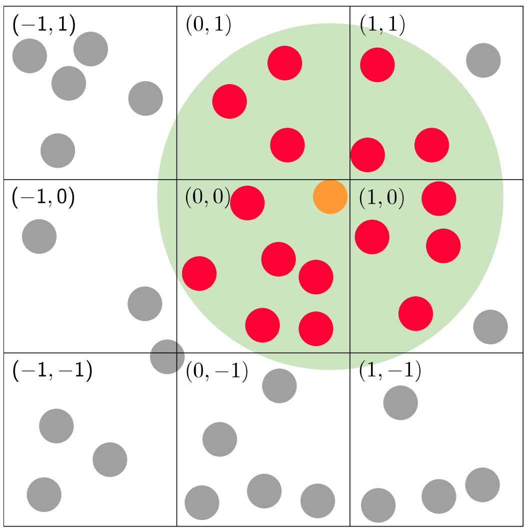


Figure 2.1: The orange particle searches all other particles, but only the ones within range are stored in the neighbour list.

2.3 Calculation speedups

By applying the linked cell list method naively, the unbonded forces between a pair of particles will be evaluated twice. By exploiting Newtons third law, we may get away with calculating the forces between a pair only once, thereby speeding up the force calculations by a factor of 2. Therefore, instead of letting the particles of each cell interact with particles located in all of the 26 neighbouring cells, they will only need to interact with half of the surrounding cells. This is accomplished by applying a stencil to each cell, as pictured in figure 2.2.

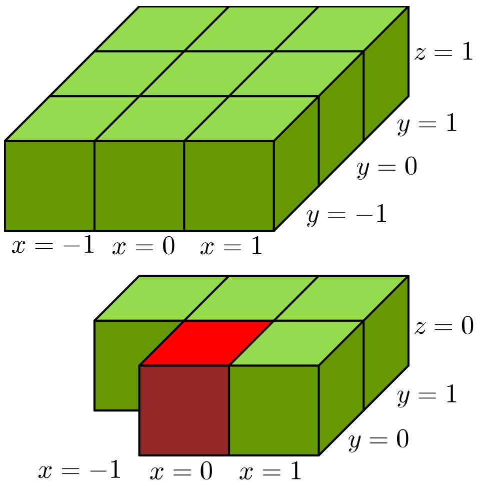


Figure 2.2: The stencil is applied by computing the neighbours between the red and the 13 green cells. Coordinates are relative to the red cell. Neighbours (i, j) inside the red cell are calculated for particles $i < j$.

2.3.1 Speeding up the neighbour search algorithm

The cell list method is a great method to reduce the search for particle neighbours to be linearly scaled. The downside is that the algorithm still needs to search through a lot of particle pairs which are further apart than the cutoff distance. In order to approximate the efficiency of the cell list algorithm, one compares the volume in which particles must be located in order to be a neighbour of another particle, namely a sphere with radius r_c, to the total volume of the 27 boxes.

$$\frac{V_{sphere}}{V_{boxes}} = \frac{4/3\pi r_c^3}{(3r_c)^3} = \frac{4\pi}{81} \approx 0.16 \tag{2.2}$$

The distance between particles r_{ij}^2 is needed to be calculated for all particles in order to determine whether a particle pair is a neighbour of another. Approximately 84% of these calculations result in a discarded particle pair, when assuming that the particle density ρ_p does not vary much in space. A lot of overhead is caused by this method, and it is therefore obvious that there may be a way to optimize this implementation further.

A straightforward solution would be to reduce the size of the cells, so that the number of potential neighbours a particle needs to search, is reduced. The neighbour search will now not only be calculated between neighbouring cells, but also cells further apart. However fewer particles are searched in total, as many of the cells previously located within the search volume may be discarded. A list of which cells that are within range of each other is precalculated, see figure 2.3. These cells are then searched when interatomic forces are being calculated.

Algorithm 2 Gonnet's algorithm

```
 1: $\mathbf{d} = \mathbf{R}_b - \mathbf{R}_a$
 2: $cutoff = |\mathbf{d}|^2 r_c^2$
 3: for $p \in list_a$ do
 4:     $dot_a^p = \mathbf{r}_p \cdot \mathbf{d}$
 5: end for
 6: for $q \in list_b$ do
 7:     $dot_b^q = \mathbf{r}_q \cdot \mathbf{d}$
 8: end for
 9: Sort $list_b$ with respect to $dot_b$
10: for $p \in list_a$ do
11:     for $q \in list_b$ do
12:         if $(dot_b^q - dot_a^p)^2 < cutoff$ then
13:             add q to p's neighbour list if $|\mathbf{r}_p - \mathbf{r}_q| \leq r_c$.
14:         else
15:             break inner for loop
16:         end if
17:     end for
18: end for
```

2.3.2 Gonnet's algorithm

Another approach which is used to reduce the overhead caused by the amount of unnecessary distance calculations, is a method suggested by Gonnet [4]. This method was shown by [5] to be more efficient than the method of smaller cells described above, and is the method we will stick to when implementing the simulation procedure.

Gonnet's algorithm is based upon projecting particles onto a vector **d** which connects the cell centres. Given a particle i which is located in the centre cell, and another particle j located in one of the neighbouring cells, with positions $\mathbf{r}_i$ and $\mathbf{r}_j$ respectively, results in the following expression when projecting the positions onto the vector **d**.

$$(\mathbf{r}_i - \mathbf{r}_j) \cdot \frac{\mathbf{d}}{d} = \cos\theta \, |\mathbf{r}_i - \mathbf{r}_j| \left| \frac{\mathbf{d}}{d} \right| \tag{2.3}$$

Rearranging the terms and squaring both sides yields

$$(\mathbf{d} \cdot \mathbf{r}_i - \mathbf{d} \cdot \mathbf{r}_j)^2 = d^2 \cos^2\theta \, (\mathbf{r}_i - \mathbf{r}_j)^2 \tag{2.4}$$

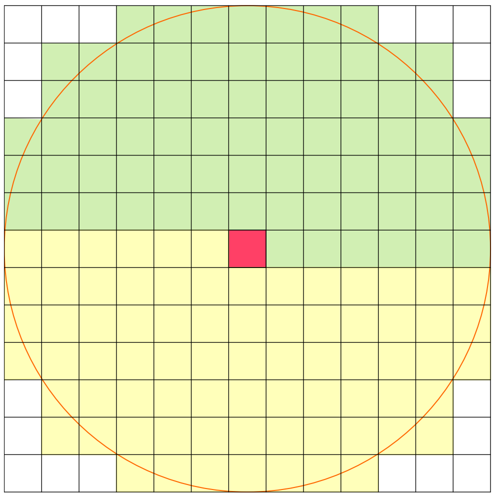


Figure 2.3: Particles within the red cell search for neighbours in the green cells. Yellow cells will search for neighbours in the red. White cells are discarded.

A list is created which contains particles of one of the neighbouring cells, and it is sorted by the product $\mathbf{d} \cdot \mathbf{r}_q$. By taking advantage of $\cos^2 \theta \leq 1$, the following inequality results from equation (2.4).

$$(\mathbf{d} \cdot \mathbf{r}_i - \mathbf{d} \cdot \mathbf{r}_j)^2 \leq d^2 \left| \mathbf{r}_i - \mathbf{r}_j \right|^2 \tag{2.5}$$

As particles will only interact with other particles within the distance r_c, we get the following expression for the upper boundary on the difference in the projections

$$(\mathbf{d} \cdot \mathbf{r}_i - \mathbf{d} \cdot \mathbf{r}_j)^2 \leq d^2 r_c^2 \tag{2.6}$$

This inequality ensures that if the differences in the projections are larger than the cutoff value, then the distance between them is also larger than the cutoff.

The sorted list is traversed when finding a potential neighbour. When the inequality (2.6) fails, the search is stopped, thereby eliminating all distance comparisons between particles that do not satisfy this inequality. The algorithm is further described in Algorithm 2 and visualized in figure 2.4.

According to Gonnet, this algorithm results in an approximate 30% speed-up of the neighbour list creation procedure. The sorting is implemented by an insertion sort algorithm, which resulted in the best speed-up when testing this algorithm compared to quicksort. Insertion sort is usually preferred when sorting small lists, which is the case here, as number of particles in each cell is usually

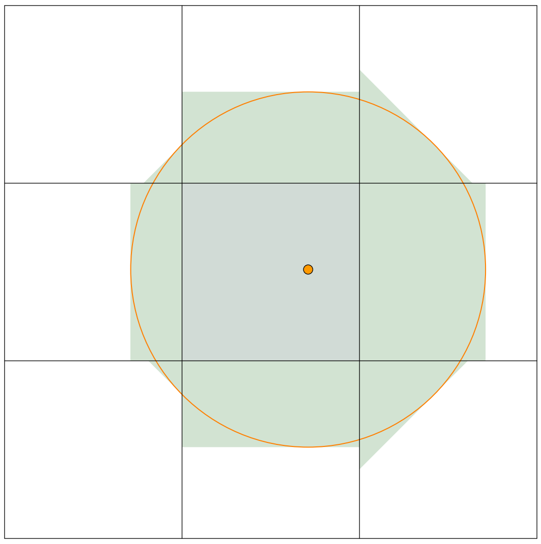


Figure 2.4: The particle only needs to search through the marked space for neighbours in Gonnet's algorithm.

about 20 for water with a density of 1 g/cm at 300 K, and even fewer for silica with a density of 2.2 g/cm. This preference is due to the fact that insertion sort has little overhead compared to quicksort.

2.4 Periodic boundary conditions

Ideally we would want to simulate a system with infinite size. This is however not possible, but it is theoretically possible to simulate very large systems. The only two obstacles are the amount of memory required to store particle information, and the computational power needed to simulate the particles movement.

A system of SiO_2 of size 1mm^3 contains 6.59×10^{22} particles. Storing the information of the positions as a double requires 1.58×10^9TB. With modern computers this is just not possible.

A way to approximate an infinite system is to introduce periodic boundary conditions. This will entail that when a particle exits the simulation domain, it reappears on the other side.

By the effects of the periodic boundary condition, particles will have to interact with an infinite number of atoms. This can however be avoided by using the minimum image convention. It states that particles will only interact with particle images which are located closest to itself. This limits interaction ranges to half a system size in each direction.

2.5 Parallelization

The parallel program is implemented by partitioning the simulation domain into $P = P_x P_y P_z$ subdomains, where P_d is the number of nodes in dimension d, and P is the total number of processor nodes. Given a processor with number p, its coordinates are (p_x, p_y, p_z) which is given by the relation $p = p_x + P_x(p_y + P_y p_z)$. A local coordinate system is defined for each processor, with origin $\mathbf{R}_p = \hat{\imath} p_x L_x/P_x + \hat{\jmath} p_y L_y/P_y + \hat{k} p_z L_z/P_z$. This results in local particle coordinates

$$\mathbf{r}_{local} = \mathbf{r}_{global} - \mathbf{R}_p \tag{2.7}$$

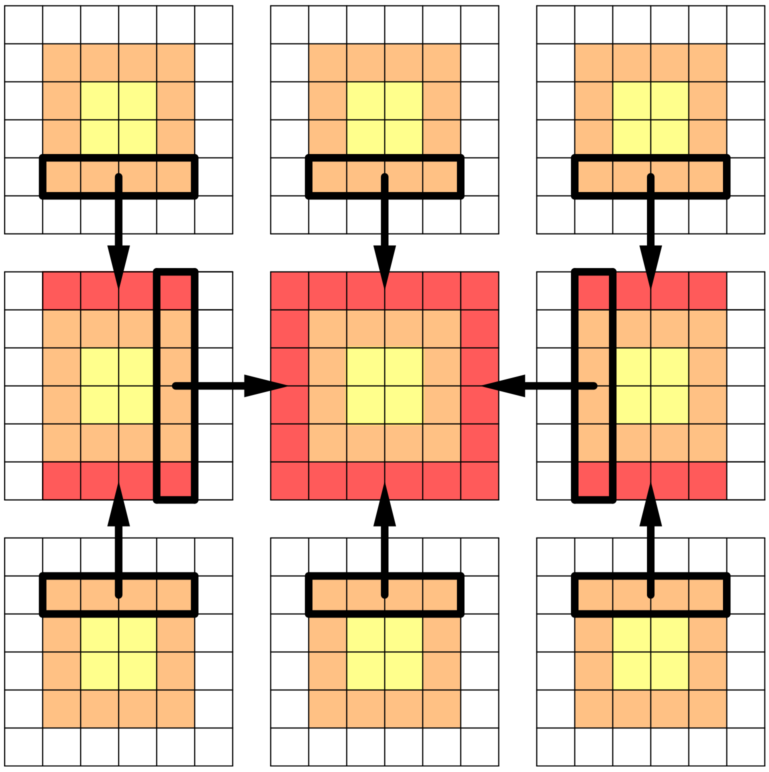


Figure 2.5: A 2D representation of the communication procedure. Particles located on the edge of a node are communicated first in the y direction and then in the x direction. This ensures that communication with only 6(4 in 2D) neighbouring nodes are needed, instead of all 26(8 in 2D). The halo(red) consist of particles located on the boundary(orange) of other nodes. Particles located in the yellow regions do not need to be communicated.

The benefit of introducing local coordinate systems, is the fact that the implementation of periodic boundary conditions and the minimum image convention, becomes rather trivial.

Particles that moves from one subdomain to another have their coordinates shifted accordingly. This shift of coordinates is independent of the subdomains. This means that the introduction of periodic boundaries are trivial, as the process is the same for shifting a particle from the first subdomain to the second, as shifting it from the last subdomain to the first. The only thing needed to perform this, is to set the last processor in a direction to be neighbour of the first processor in that direction, and vice versa.

The partitioning into local subdomains complicates the previously discussed neighbour search, as some particles which are on the edge of the subdomain of a processor, will need to know about particles on the edge of the subdomain of a neighbouring processor. Particles will therefore need to be communicated between processors when calculating the interatomic forces. The communication procedure will first send particles which have exited one subdomain and entered another during the integration procedure. Then the particles on the border will be communicated to neighbouring nodes. A way to maximize the efficiency of the communication is illustrated in figure 2.5.

Also the way the stencil is used in the neighbour search procedure must be modified. There is no need to find neighbours between particles which are located on the halo, as the forces between them will be calculated in a different processor node. But neighbours between the local domain and the halo must be found. This is done as illustrated in Algorithm 3.

Algorithm 3 Stencil applied on a parallel system

```
1: for $C_l \in$ local cells do
2:     for $C_s \in$ stencil($C_s$) do
3:         Find neighbours between $C_l$ and $C_s$.
4:     end for
5: end for
6: for $C_h \in$ halo do
7:     for $C_s \in$ stencil($C_s$) do
8:         if $C_s \in$ local cells then
9:             Find neighbours between $C_l$ and $C_s$.
10:        end if
11:    end for
12: end for
```

2.6 Time integration

The purpose of a molecular simulation is to begin with a set of initial particle states $\{\mathbf{r}_i{}^0, \mathbf{v}_i{}^0\}$, then apply Newton's equations of motion using the forces given from a desired potential, which results in particle states at a different time $\{\mathbf{r}_i{}^n, \mathbf{v}_i{}^n\}$.

Applying Newtons equations of motion results in a set of differential equations which are too complex to solve analytically. An approach which works well is to use a numerical finite difference scheme for the time integration. Some thought must be put into which integration method to use. Some methods, such as the forward Euler scheme, is a scheme which is susceptible to high energy drifts, and should therefore not be used. Other schemes such as the Runge-Kutta method, which is of order $\mathcal{O}(\Delta t^4)$, are too costly. It is usually only used in special situations, such as when the energies of the system result in relativistic velocities.

In our implementation we will use methods which are not too costly. Forces are only computed once every timestep. And accuracy of the integration schemes are sufficient, being of order $\mathcal{O}(\Delta t^2)$.

2.6.1 Velocity-Verlet

A more sophisticated algorithm than the forward Euler scheme, which is called the velocity-Verlet algorithm [6], is not susceptible to energy drifts in the long run. It can be derived by looking at the Taylor-expansion of the position of a particle $\mathbf{r}$, its velocity $\mathbf{v}$ and its acceleration $\mathbf{a} = \mathbf{F}/m = -\nabla U/m$ at the time t.

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + \frac{\Delta t^2}{2}\mathbf{a}(t) + \mathcal{O}(\Delta t^3) \tag{2.8a}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \Delta t\mathbf{a}(t) + \frac{\Delta t^2}{2}\dot{\mathbf{a}}(t) + \mathcal{O}(\Delta t^3) \tag{2.8b}$$

$$\mathbf{a}(t+\Delta t) = \mathbf{a}(t) + \Delta t\dot{\mathbf{a}}(t) + \mathcal{O}(\Delta t^2) \tag{2.8c}$$

Rearranging (2.8c) and combining it with (2.8b) gives the equation

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{\Delta t}{2}[\mathbf{a}(t+\Delta t) + \mathbf{a}(t)] + \mathcal{O}(\Delta t^3) \tag{2.9}$$

By introducing an intermediate velocity $\mathbf{v}(t+\Delta t/2)$, equation (2.8a) may be expressed as

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t+\Delta t/2) + \mathcal{O}(\Delta t^3) \tag{2.10}$$

The accelerations at the new timestep may now be calculated using this set of positions. By the use of the newly calculated accelerations we may finally find the velocities at the new timestep. The order in which the different quantities are computed, is listed in equations (2.11)

$$
\begin{aligned}
\mathbf{v}(t+\Delta t/2) &\simeq \mathbf{v}(t) + \frac{\Delta t}{2}\mathbf{a}(t) && \text{(2.11a)}\\
\mathbf{r}(t+\Delta t) &\simeq \mathbf{r}(t) + \Delta t\mathbf{v}(t+\Delta t/2) && \text{(2.11b)}\\
\mathbf{a}(t+\Delta t) &= \mathbf{a}(\mathbf{r}(t+\Delta t)) && \text{(2.11c)}\\
\mathbf{v}(t+\Delta t) &\simeq \mathbf{v}(t) + \frac{\Delta t}{2}\mathbf{a}(t+\Delta t) && \text{(2.11d)}
\end{aligned}
$$

The local error is as stated above $\mathcal{O}(\Delta t^3)$. By summing up the error terms, the resulting global error will be of order $\mathcal{O}(\Delta t^2)$.

2.6.2 Crank-Nicolson

A method which is widely used in numerical analysis is the Crank-Nicolson(CN) integration scheme. It is a midpoint scheme, where the derivative is evaluated at the midpoint of the time interval of length Δt. Just like velocity Verlet scheme, the CN method scales quadraticly as $\mathcal{O}(\Delta t^2)$.

Given a general first order ODE $\dot{\psi} = G(t, \psi)$, the Crank-Nicolson disctretization gives the expression

$$\frac{\psi^{n+1} - \psi^n}{\Delta t} = G\left[\psi^{n+1/2}, (n+1/2)\,\Delta t\right] = G^{n+1/2} \tag{2.12}$$

As the value of G is usually not known at the midpoint, the right hand size of (2.12) is approximated using some sort of averaging function $G^{n+1/2} \simeq A(t, \psi^n, \psi^{n+1})$. This eliminates the dependency on the midpoint value, and makes it possible to express the equation in terms of the known ψ^n and the unknown ψ^{n+1}.

In order to maintain the scaling of $\mathcal{O}(\Delta t^2)$, this average function has to at least scale quadratically. The arithmetic and geometric mean are two examples of averaging functions which both scale quadratically, and are often used in combination with the CN scheme.

If possible, A is chosen as a linear function of ψ^{n+1}. If a linearization is not possible, methods such as Picard iteration or Newtons method can be applied in order to find the values of ψ.

By assuming (that) A can be expressed as a linear function of ψ^{n+1}, equation (2.12) becomes

$$\frac{\psi^{n+1} - \psi^n}{\Delta t} = A\left(G^n, G^{n+1}\right) = a^n\psi^{n+1} + b^n \tag{2.13}$$

Here a^n and b^n can be dependent on both the time t and the value ψ^n. Modifying this equation yield the expression for ψ^{n+1}

$$\psi^{n+1} = \frac{\psi^n + \Delta t b^n}{1 - \Delta t a^n} \tag{2.14}$$

By applying this scheme on the motion of N particles, it would return N nonlinear equations which all are dependent on each other. This is due to the severely non-linear nature of interatomic potentials, which means no function A can be found which returns a linear equation of ψ^n. It is therefore not feasible to use CN for integrating particle trajectories, and we will rather use the velocity-Verlet method for this purpose. The CN method is however useful when we want to integrate other quantities which are involved in the simulation, such as the friction coefficients of the Nosé-Hoover thermostat.

Chapter 3

Theory of molecular dynamics

Atoms are usually modelled as point particles, and their trajectories are calculated using Newton's equations of motion. By using this approximation, we may find the positions $\mathbf{r}_i$, velocity $\mathbf{v}_i$ and interatomic forces $\mathbf{F}_i$ for all particles in the system at a given time. From those values, thermodynamical properties of the system, such as temperature and pressure, may be found.

In order to calculate particle trajectories, the forces acting upon a particle must be found. These are found using an interatomic potential model, also called a force field model. Forces acting on particle i are found by the following equation.

$$m_i\mathbf{a}_i = \mathbf{F}_i = -\nabla_i V(\{\mathbf{r}_j\}) \tag{3.1}$$

where $\{\mathbf{r}_j\}$ is the collection of the positions of all of the particles in the system, and

$$\nabla_i = \frac{\partial}{\partial \mathbf{r}_i} = \hat{\imath}\frac{\partial}{\partial x_i} + \hat{\jmath}\frac{\partial}{\partial y_i} + \hat{k}\frac{\partial}{\partial z_i} \tag{3.2}$$

There have been developed several different potentials which models different kinds of particle behaviour. Some, like the Lennard-Jones potential, are only capable of modelling neutral particles like argon. Others may model complex chemistry, like the ReaxFF model. This thesis focuses on the ClayFF model which has been shown to be able to model the mechanics of silica structures well, and the USC model which is capable of modelling simple chemical processes between silica and water.

3.1 Statistics sampling

A molecular dynamic simulation produces microscopic details about the system, the positions and velocities of all particles are known in detail. This is not the case when performing a real experiment, where our results are in the form of time averages of thermodynamical properties. In order to convert our microscopic details about a simulated system into physically understood quantities, the theory of statistical mechanics is used.

3.1.1 Ergodicity

A systems phase space coordinate is said to be its collection of particle positions and momentums $\mathbf{\Gamma} = \{\mathbf{r}(t), \mathbf{p}(t)\}$. For a system of N particles, the dimension of its phase space is $6N$.

There are a massive amount of phase coordinates which share macroscopic observable properties. The collections of these states are called an ensemble. The macroscopic properties are the set of observable quantities(energy, temperature, diffusion coefficient etc.). These are not dependent on the details of each particle, but rather the state of the system as a whole.

The probability of a system being at a phase space coordinate $\mathbf{\Gamma}$ is given by the distribution $\rho(\mathbf{\Gamma})$. An observable macroscopic property A_{obs}, will then be determined by the instantaneous values A at the phase coordinates $\mathbf{\Gamma}$ by

$$A_{obs} = \int_{\mathbf{\Gamma}} A(\mathbf{\Gamma})\rho(\mathbf{\Gamma})d\mathbf{\Gamma} \tag{3.3}$$

This is called the ensemble average. It is not easily calculated, as the probability distribution ρ is generally not known. We may avoid calculating ρ completely, by taking advantage of the ergodicity hypothesis. This hypothesis states that over a long period of time, the time spent within a region of a phase space coordinate is proportional to the probability density at that coordinate. Equation (3.3) then becomes

$$A_{obs} = \lim_{t\to\infty} \frac{1}{t} \int_0^t A(\mathbf{\Gamma}(t'))\mathrm{d}t' \tag{3.4}$$

Naturally it's impossible to average over infinite time when simulating a system using a computer. A computational approximation is to average over a sufficient large time interval t_{obs}.

3.1.2 Ensembles

A closed molecular system, which is not interacting in any way with the outside world, is said to be in the microcanonical ensemble, or NVE. This means that the number of particles in the system, the volume, and the energy is kept constant throughout the simulation. Another ensemble, called the canonical ensemble(NVT), is when the temperature, rather than energy is held constant. This is usually the case for physical experiments, and is therefore often used when sampling statistics from a molecular dynamical simulation. In order to achieve a constant temperature, a so-called thermostat has to be applied, which regulates the temperature, and thereby the velocities of the particles. For more information on thermostats see chapter 4.

3.2 Temperature

The instantaneous temperature of a system is calculated using the equipartition theorem $E_K = (f/2)k_BT$. Where E_K is the kinetic energy of the system, f is number of degrees of freedom, T is the temperature and k_B is Boltzman's constant. For a system of N particles, the number of degrees of freedom is $3N$. As suggested by the previous section, temperature should be averaged over time in order to achieve a thermodynamical representation of the temperature, which gives us

$$T = \frac{2\langle E_K \rangle}{3Nk_B} \tag{3.5}$$

3.3 Diffusion

Diffusion is the process which causes a drop of a fluid to be dissolved in another medium in the absence of flow. The process is caused by the fact that molecules in a fluid have motion, which causes them to travel around in an irregular pattern as they interact and collide with each other. The diffusion process is modelled macroscopically by Fick's law

$$\mathbf{j} = -D\nabla c \tag{3.6}$$

where $\mathbf{j}$ is diffusion flux, c is the concentration of the diffusive fluid, and D is a proportionality constant called the diffusion coefficient. We want to find the self-diffusion, which is defined as the diffusion of a species of particles among identical solvent particles. The self-diffusion takes the form of the Einstein relation, given by

$$6D = \lim_{t\to\infty} \frac{\mathrm{d}\langle r^2(t)\rangle}{\mathrm{d}t} \tag{3.7}$$

Here $\langle r(t)^2\rangle$ is the particle average of the square of the distance travelled. By finding the diffusion coefficient after a sufficient amount of time, we will be able to determine whether the matter studied is a solid or liquid. The diffusion coefficient will be zero for solids, as particles are nearly frozen in place and only vibrates about a point. For liquids it will grow larger with temperature, as the particles are travelling freely, and increases as the particles kinetic energy grows larger with temperature. The phase transition from solid to liquid is then found by determining the point where the diffusion coefficient starts being non-zero.

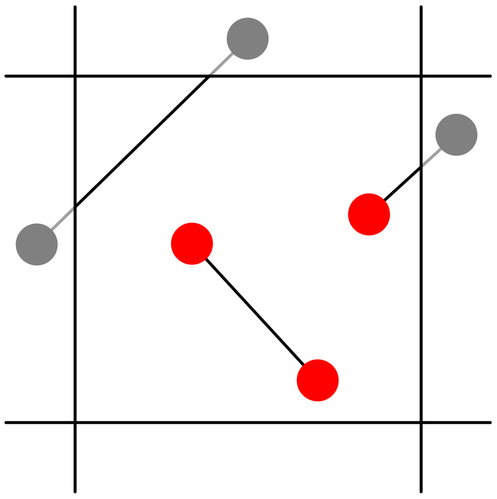


Figure 3.1: There are three possibilities for particle positions, when calculating the line segment ratio l_{ij}. Both particles may be inside the cell ($l_{ij} = 1$), one particle may be inside and the other outside the cell, or both are outside the cell.

3.4 Pressure calculation

The pressure P of an ideal gas in a volume V can be calculated using the ideal gas law: $PV = NTk_B$. By definition particles in an ideal gas do not interact with each other. By introducing the complexity of particle interactions, the calculation of pressure require an addition term, called the virial. The virial is a quantity which relates the average total kinetic energy to the total potential energy. It is usually calculated by only taking two body interactions into account.

By including the virial, the expression for the pressure consisting of interacting particles will be

$$P = \underbrace{\frac{NTk_B}{V}}_{\text{ideal gas}} + \underbrace{\frac{1}{3V}\sum_{i<j} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij}}_{\text{virial term}} \tag{3.8}$$

This expression finds the global pressure of the system at an instantaneous time t. In many cases it may be more interesting to know the pressure as a function of position, usually when averaged over a time period. Cormier et. al. [12] devised a method which made it possible to find the pressure at discretised points in space. In order to be able to do this, we need to define a set of cells which partition the system volume. In total there are $N_x \times N_y \times N_z$ cells, each with a size of $L_x/N_x \times L_y/N_y \times L_z/N_z$. We define $\Omega_\mathbf{r}$ to be the cell in which the point $\mathbf{r}$ is located. By the use of this definition, it is possible to find the value of the local instantaneous pressure of the cells. The pressure of the cell located at the point $\mathbf{r}$ will be given by a similar formula as the total pressure of the system, with just one modification in each term:

$$\mathcal{P}(\mathbf{r}) = \frac{1}{3V_\Omega}\left[\sum_i m_i \mathbf{v}_i^2 \Lambda_i(\mathbf{r}) + \sum_{i<j} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} l_{ij}(\mathbf{r})\right] \tag{3.9}$$

where V_Ω is the volume of the cell and

$$\Lambda_i(\mathbf{r}) = \begin{cases} 1, & \mathbf{r}_i \in \Omega_\mathbf{r} \\ 0, & \mathbf{r}_i \notin \Omega_\mathbf{r} \end{cases} \tag{3.10}$$

$l_{ij}(\mathbf{r})$ is a function which is more complex to evaluate. It is given by the portion of the line segment connecting particle i and j which is contained in $\Omega_\mathbf{r}$. Note that particles which are not contained in a cell may add to the value of the cells pressure via the virial term. This is due to the line connecting the two particles may cross the cell, as depicted in figure 3.1.

3.5 Contact mechanics of a cylinder

In contact dynamics, deformation of solids which are in contact with each other are studied. In this thesis we will be studying a cylinder in contact with a so called half-space. A half space is a structure which consists of all points located in the space behind a two dimensional plane.

According to Johnson[20], two cylinders in contact with parallel axis will exert a pressure in the contacting area which takes the following expression

$$p(r) = p_0\sqrt{1 - \frac{r^2}{a^2}} \tag{3.11}$$

Here a is twice the width of the contact area, which is called the contact radius. r is distance from the centre of the contact area, perpendicular to the cylinders axis. p_0 is the maximum pressure, located in the middle of the contact area. This pressure is can be determined by other physical properties, by the relation

$$p_0 = \frac{aE^*}{2R} \tag{3.12}$$

Here R is the curvature defined as $1/R = 1/R_1 + 1/R_2$, where R_1 and R_2 are the radii of the two cylinders in contact. E^* is the so-called reduced elastic modulus. This is a material property which is dependant on the material of the two objects in contact.

By letting R_2 tend towards infinity, the second cylinder will be a half space. We may therefore use the previous equations when calculating contact properties of a cylinder in contact with a half space, by applying $R = R_1$.

Chapter 4

Thermostats

In order to determine the physical properties of a system, one needs to know its temperature. It is also nearly always desired to perform an experiment where the system has a preselected temperature. Often when a simulation is started with a newly initialized molecular system, its temperature will fluctuate and change drastically during the first period of the simulation. It is nearly impossible to predict which temperature a system will reach based on its initial state alone. Therefore a method to force the system to reach a given temperature, is a valuable tool.

It is also desired to have a method to keep a systems temperature from changing much during the experiment, which is often the case of physically performed experiments. This also ensures that the accumulation of numerical errors will not cause the systems energy to drift in the long run, as energy drifts on large timescales will be managed by the thermostat. In the following sections, two methods will be presented which is used to manipulate the temperature of a molecular system. The Berendsen thermostat, discussed in section 4.1, and the Noseé-Hoover thermostat discussed in section 4.2 and 4.3.

4.1 Berendsen thermostat

The Berendsen thermostat [15] is a very efficient method to force a system to quickly reach a desired temperature.

The method is defined by introducing a so-called weak coupling between an external heat bath and the system, where the heat bath has the desired temperature T_0 which is fixed in time. The weak coupling results in equation (4.1).

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{\tau}(T_0 - T) \tag{4.1}$$

Solving it for $T(t)$ results in a function which decays to the desired temperature

$$T(t) = Ce^{-t/\tau} + T_0 \tag{4.2}$$

Here τ is a parameter which determines how tight the heat bath and the system are coupled together, and therefore also determines the rate in which the system reaches the desired temperature.

Approximating the differential in equation (4.1) using a forward Euler method, gives an approximation of the temperature T_{new} after the timestep.

$$\frac{T_{new} - T}{\Delta t} = \frac{1}{\tau}(T_0 - T) \tag{4.3}$$

$$\frac{T_{new}}{T} = 1 + \frac{\Delta t}{\tau}\left(\frac{T_0}{T} - 1\right) \tag{4.4}$$

By taking advantage of the equipartition theorem, one can rewrite the temperature fraction in equation (4.4) as

$$\frac{T_{new}}{T} = \frac{K_{new}}{K} = \frac{\sum_i m_i|\mathbf{u}_i|^2}{\sum_i m_i|\mathbf{v}_i|^2} \tag{4.5}$$

The easiest way to enforce the energy scaling is through the linear scaling of the velocities $\mathbf{u}_i = \gamma\mathbf{v}_i$, which results in the scaling factor being determined by the relation in equation (4.6), and the updated velocities are $\mathbf{u}_i = \gamma\mathbf{v}_i$.

$$\gamma^2 = 1 + \frac{\Delta t}{\tau}\left(\frac{T_0}{T} - 1\right) \tag{4.6}$$

This procedure will, as mentioned, force the system to quickly reach a desired temperature. It makes however the system behave slightly non-physical, and should therefore not be used while sampling statistics.

Notice that if we set $\tau = \Delta t$, this would be the same as resetting the temperature of the system to the desired temperature at each timestep.

4.2 Nosé-Hoover Thermostat

A way of controlling a systems temperature, and still being able to sample physical statistics from it, is to apply the so-called Nosé-Hoover thermostat. It is a method which is based upon a modified hamiltonian of the molecular system. The new Hamiltonian takes the form [6, p. 498] [13]

$$\mathcal{H} = \sum_i \frac{m_i \mathbf{v}_i^2}{2s^2} + V(\mathbf{r}) + \frac{1}{2}Q\dot{s}^2 + \frac{3}{2}NkT \ln s \tag{4.7}$$

s is a dynamical variable which has an associative mass Q. The modification of the hamiltonian will in turn affect the equations of motion. By introducing the so-called thermodynamic friction coefficient $\zeta = \dot{s}$, the Nosé-Hoover equations of motion can be expressed as follows

$$\dot{\mathbf{r}}_i = \mathbf{v}_i \tag{4.8a}$$
$$\dot{\mathbf{v}}_i = \mathbf{a} - \zeta \mathbf{v}_i \tag{4.8b}$$
$$\dot{\zeta} = \frac{1}{Q}\left(\frac{1}{2}\sum_i m_i \mathbf{v}_i^2 - \frac{3}{2}Nk_B T_0\right) \tag{4.8c}$$

Here T_0 is the desired temperature, and Q is a relaxation parameter which determines the coupling of the heat bath to the system.

It's not as straight forward to apply this method to the simulation procedure, as it was with the Berendsen thermostat. The equations of motion are changed, and therefore the velocity Verlet integration process needs a modification as well. Two of the three equations 2.11, can be used as is, by substituting the force with the added friction term. The third one 2.11d, which updates the velocity to the new timestep, must be modified.

$$\mathbf{v}_i^{n+1/2} = \mathbf{v}_i^n + \frac{\Delta t}{2}\left[\mathbf{a}^n - \zeta^n \mathbf{v}_i^n\right] \tag{4.9a}$$
$$\zeta^{n+1/2} = \zeta^n + \frac{\Delta t}{2Q}\left[\frac{1}{2}\sum_i m_i (\mathbf{v}_i^n)^2 - \frac{3}{2}Nk_B T_0\right] \tag{4.9b}$$
$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^{n+1/2} + \frac{\Delta t}{2}\left[\mathbf{a}^{n+1} - \zeta^{n+1}\mathbf{v}_i^{n+1}\right] \tag{4.9c}$$
$$\zeta^{n+1} = \zeta^{n+1/2} + \frac{\Delta t}{2Q}\left[\frac{1}{2}\sum_i m_i (\mathbf{v}_i^{n+1})^2 - \frac{3}{2}Nk_B T_0\right] \tag{4.9d}$$

Equation (4.9a) and (4.9b) can be calculated as is, however equation (4.9c) and (4.9d) are a coupled set of equations and needs more advanced techniques in order to find a solution. Implicit methods such as a predictor corrector scheme or a solution by iteration have been used to solve these equations [17]. In this thesis we will take the approach of using the Crank-Nicolson integration scheme, as suggested by Bond et. al. [16] which produces a set of equations which is explicit solvable. This is a method which scales as $\mathcal{O}(\Delta t^2)$, which is equally well as the Verlet integration scheme itself, and will therefore not introduce additional numerical errors.

Using the CN scheme, the expression for the updated friction coefficient becomes

$$\zeta^{n+1} = \zeta^n + \frac{\Delta t}{2Q}\left[\sum_i m_i \left(\mathbf{v}_i^{n+1/2}\right)^2 - 3Nk_BT_0\right] \tag{4.10}$$

The updated velocity at the new timestep can now be found. After some manipulation of equation (4.9c) it becomes

$$\mathbf{v}_i^{n+1} = \frac{2\mathbf{v}_i^{n+1/2} + \Delta t\mathbf{a}_i^{n+1}}{2 + \Delta t\zeta^{n+1}} \tag{4.11}$$

It is now possible to derive the Noseé-Hoover integration procedure, which is given in algorithm 4.

Algorithm 4 Nose-Hoover integration procedure

```
for $t \in time$ do
    for $p \in Particles$ do
        Find $\mathbf{v}_i^{n+1/2}$ by applying equation (4.9a)
        Update $\mathbf{r}_i$ by applying equation (2.11b)
    end for
    Update $\zeta$ by applying equation (4.10)
    Update forces
    for $p \in Particles$ do
        Find $\mathbf{v}_i^n$ by applying equation (4.11)
    end for
end for
```

Great care must be taken when choosing a value for the parameter Q. A too small value may cause instabilities, because of equations 4.9a and 4.9c may cause particles to suddenly change direction if ζ grows too big. A value which is too large may however cause unphysical oscillations in temperature. It was suggested by Martyna et. al. [14] to use the value $Q = Nk_BT\tau^2$, where $\tau \geq 20\Delta t$ is a time scaling factor, similar to τ in the Berendsen thermostat.

4.3 Nosé-Hoover Chain

It was shown by Noseé [13] that the standard Noseé-Hoover thermostat approximates the canonical ensemble very well in most cases. One of the main assumption was the fact that the dynamics of the system had to be ergodic. This is not the case for stiff systems, or systems where external forces are applied. Martyna et. al. [14] suggested a method which solves this problem by introducing multiple thermostats applied in a chain. This would be equivalent to thermalizing the imaginary particles introduced by the standard Nosé-Hoover method by another heatbath of imaginary particles. The procedure is then applied to the newly introduced heatbath, and the chain is continued for as long as desired. As only the first thermostat affects the particles directly, the introduced degrees of freedom increase the computational cost by negligible small amount, compared to the computation required to integrate the particles motion.

The thermostat takes the form as described in equations (4.12) when given a chain of length M.

$$\dot{\mathbf{v}}_i = \mathbf{a} - \mathbf{v}_i\zeta_0 \tag{4.12a}$$

$$\dot{\zeta}_0 = \frac{1}{Q_0}\left[\sum \frac{1}{2}m_i\mathbf{v}_i^2 - \frac{3}{2}NkT\right] - \zeta_0\zeta_1 \tag{4.12b}$$

$$\dot{\zeta}_j = \frac{1}{Q_j}\left[\frac{1}{2}Q_{j-1}\zeta_{j-1}^2 - \frac{1}{2}kT\right] - \zeta_j\zeta_{j+1} \tag{4.12c}$$

$$\zeta_M = 0 \tag{4.12d}$$

This set of equations are non-linearly coupled, and will therefore be difficult to solve as is. In order to find an easily solvable expression, we start by discretizing the equations using the Crank-Nicolson scheme, just as we did with the standard Nosé-Hoover equations. The resulting discretized equations will become

$$\frac{\zeta_0^{n+1} - \zeta_0^n}{\Delta t} = \frac{1}{2Q_0}\left[\sum m_i\left(\mathbf{v}_i^{n+1/2}\right)^2 - 3NkT\right] - \zeta_0^{n+1/2}\zeta_1^{n+1/2} \tag{4.13a}$$

$$\frac{\zeta_j^{n+1} - \zeta_j^n}{\Delta t} = \frac{1}{2Q_j}\left[Q_{j-1}(\zeta_{j-1}^{n+1/2})^2 - kT\right] - \zeta_j^{n+1/2}\zeta_{j+1}^{n+1/2} \tag{4.13b}$$

In order to linearize these equations, the mixed geometric-arithmetic mean is utilized.

$$\zeta_i^{n+1/2}\zeta_j^{n+1/2} = \frac{1}{2}\left(\zeta_i^n\zeta_j^{n+1} + \zeta_i^{n+1}\zeta_j^n\right) + \mathcal{O}(\Delta t^2) \tag{4.14}$$

In the case where $i = j$, it will be reduced to the ordinary geometric mean. By applying this averaging function, it will be possible to linearize equations (4.13).

$$\frac{\zeta_0^{n+1} - \zeta_0^n}{\Delta t} = \frac{1}{2Q_0}\left[\sum m_i \left(\mathbf{v}_i^{n+1/2}\right)^2 - 3NkT\right] - \frac{1}{2}\left(\zeta_0^n \zeta_1^{n+1} + \zeta_0^{n+1}\zeta_1^n\right) \tag{4.15a}$$

$$\frac{\zeta_j^{n+1} - \zeta_j^n}{\Delta t} = \frac{1}{2Q_j}\left(Q_{j-1}\zeta_{j-1}^n \zeta_{j-1}^{n+1} - kT\right) - \frac{1}{2}\left(\zeta_j^n \zeta_{j+1}^{n+1} + \zeta_j^{n+1}\zeta_{j+1}^n\right) \tag{4.15b}$$

In essence, this is a set of tridiagonal system of equations.

$$A_j^n \zeta_{j-1}^{n+1} + B_j^n \zeta_j^{n+1} + C_j^n \zeta_{j+1}^{n+1} = D_j^n \tag{4.16}$$

where the coefficients are

$$A_j^n = -\frac{\Delta t Q_{j-1}\zeta_{j-1}^n}{2Q_j} \tag{4.17a}$$

$$B_j^n = 1 + \frac{1}{2}\zeta_{j+1}^n \Delta t \tag{4.17b}$$

$$C_j^n = \frac{1}{2}\zeta_j^n \Delta t \tag{4.17c}$$

$$D_j^n = \zeta_j^n - \frac{kT\Delta t}{2Q_j} \tag{4.17d}$$

$$D_0 = \zeta_0^n - \frac{\Delta t}{2Q_0}\left[3NkT - \sum_i m_i \left(\mathbf{v}_i^{n+1/2}\right)^2\right] \tag{4.17e}$$

$$B_{M-1} = 1 \tag{4.17f}$$

$$A_0 = C_{M-1} = 0 \tag{4.17g}$$

A solution of ζ_j^{n+1} is then found by the use of a tridiagonal matrix solver, like the Thomas algorithm. After finding the value of ζ_0^{n+1}, we again apply the same equation as with the standard Nosé-Hoover algorithm, in order to find the new particle velocities

$$\mathbf{v}_i^{n+1} = \frac{2\mathbf{v}_i^{n+1/2} + \Delta t \mathbf{a}_i^{n+1}}{2 + \Delta t \zeta_0^{n+1}} \tag{4.18}$$

Again, we will have to be careful when choosing the values of Q_j, as the same problems may occur with this algorithm as with the Nosé-Hoover algorithm. Martyna et. al. [14] suggested using $Q_0 = NkT\tau^2$ and $Q_j = kT\tau^2$ for $j > 0$, where τ is the same parameter as in the standard Nosé-Hoover thermostat.

Chapter 5

Potentials in molecular dynamics

The motion of particles is determined by two factors, one of which are the external forces acting on a molecular system, the other is the interaction caused by interatomic potentials. The interatomic potentials are usually developed from advanced quantum mechanical calculations, or sometimes approximated using empirical models which fit experimental data.

The most general form of the potential used in molecular dynamics can be expressed as the following sum

$$V = \sum_i V_i(\mathbf{r}_i) + \sum_{i,j} V_{ij}(\mathbf{r}_i, \mathbf{r}_j) + \sum_{i,j,k} V_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) + \dots \tag{5.1}$$

The first term V_i determines the external forces which are applied to the system. The second term V_{ij} is the two body interatomic potential term which depends on the distance vector between two particles. V_{ijk} is the three body potential which is used to calculate interior forces in molecules and forces during chemical processes. The formation of silanol groups in the interfaces between water and silica are such chemical processes in which the three-body potential plays a big role. Higher order terms are usually omitted from most models, due to the large computational capacity required to compute such forces, and the difficulty in evaluating these compared to lower order terms. In this thesis, we use models which are truncated at the three-body term V_{ijk}.

We define the two body forces acting from particle j upon i as $\mathbf{F}^i_{ij}$, which means $\mathbf{F}^j_{ij} = -\mathbf{F}^i_{ij}$. By this definition, and the fact that two body potentials are a function of distance between the interacting atoms, the force acting upon a particle i will be given as

$$\mathbf{F}^i_{ij} = -\nabla_i V_{ij}(r_{ij}) = -\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}} \nabla_i \left(\frac{1}{|\mathbf{r}_j - \mathbf{r_i}|} \right) = -\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.2}$$

For a three body potential, the case is a bit more complex, as the potential is a function of angle, and often distances as well. In general a three body potential has the form $V_{ijk} = V_{ijk}(\mathbf{r}_{ij}, \mathbf{r}_{ik}, \theta_{ijk})$. And the angle θ_{ijk} is defined to be the angle between the two vectors $\mathbf{r}_{ij}$ and $\mathbf{r}_{jk}$. The total potential contribution due to this particle triple will be $V_{ijk} + V_{jki} + V_{kij}$. However, often only one of these terms will contribute to the total energy, as the other two will be set to zero in the force field model. For instance, only the potential due to the H–O–H angle will result in an energy contribution in the water molecule in most models.

The forces acting upon a particle j(chosen for simplicity) will be expressed as follows:

$$\mathbf{F}^j_{ijk} = -\frac{\partial V_{ijk}}{\partial r_{ij}}\frac{\mathbf{r}_{ij}}{r_{ij}} - \frac{\partial V_{ijk}}{\partial \theta_{ijk}}\nabla_j \theta_{ijk} \tag{5.3}$$

where $\nabla_j \theta_{ijk}$ can be shown to reduced to

$$\nabla_j \theta_{ijk} = \frac{\mathbf{r}_{ij}\cos\theta_{ijk} - \mathbf{r}_{ik}}{r_{ij}\sin\theta_{ijk}} \tag{5.4}$$

The same expression can be found for particle k by interchanging with j. By using Newtons third law, we will find the forces acting upon particle i due to the three body potential to be $\mathbf{F}^i_{ijk} = -(\mathbf{F}^j_{ijk} + \mathbf{F}^k_{ijk})$.

5.1 Lennard Jones

One of the most widely used potentials in molecular dynamical simulations is the Lennard-Jones potential. It has been very successful of modeling non-reactive particles like noble gases [7]. It is a relatively simple potential, which takes the following form

$$V(r) = \epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]. \tag{5.5}$$

It is also common to express the LJ potential in terms of the coordinate of the minima R_0 instead of zero point σ. This results in the expression

$$V(r) = \epsilon\left[\left(\frac{R_0}{r}\right)^{12} - 2\left(\frac{R_0}{r}\right)^{6}\right]. \tag{5.6}$$

The resulting forces between two particles i and j, which interact using this potential, will be

$$\mathbf{F}_{ij} = 24\epsilon \left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right] \frac{\mathbf{r}_{ij}}{r_{ij}^2} \tag{5.7}$$

As it is a very simple two body potential, it is easily implemented and very efficient compared to more complex potentials.

5.1.1 Cutoff and tail correction

The most time consuming task of a molecular dynamical simulation is the calculation of interatomic forces. Theoretically all particles react with all other particles, no matter how far they are apart. This results in the total number of force calculations being $N(N-1)/2$, which is of order $\mathcal{O}(N^2)$. However, the strength of a particle interaction is determined by the distance between the atom pairs. Usually this is inversely proportional to a monomial of the distance, which quickly falls towards zero. This means the interactions between particles far away from each other will be minuscule compared to particles located closer together.

The value of the forces due to the Lennard-Jones potential at $r = 3\sigma$ is $|\mathbf{F}| \approx 0.0101\epsilon$. The value at the minima($r \approx 1.24$) has the value -2.4ϵ, which is nearly 240 times larger than at $r = 3\sigma$. Therefore it is reasonable to introduce a cutoff for this potential at the distance $r = 3\sigma$.

Just stopping the potential at the cutoff alone is not considered good practice, as it introduces discontinuities in the potential energy. One should rather shift the potential such that its value is zero at the cutoff distance. This should also be carried out for the resulting forces, to avoid sudden unphysical small jerks in the particle movement. To achieve this, a given two body potential V_{ij}, which has the value V_{ij}^c at the cutoff distance r_c, will be shifted according to equation (5.8)

$$V_{ij}^{shifted} = \begin{cases} V_{ij} - V_{ij}^c + (r - r_c)\left[dV_{ij}/dr\right]_{r=r_c} & r < r_c \\ 0 & r > r_c \end{cases} \tag{5.8}$$

5.2 Clay force field

Monoatomic systems can usually be reasonable well described by a two body interaction alone. More complex systems where several species of particles are involved often needs a more rigorous model. For the case of silica, this is not necessarily needed. It is however needed for the water molecule, as a three body potential is required in order to model the vibrational behaviour of hydroxides sufficiently well.

The clay force field model, is a molecular dynamical model used to model minerals, and their interfaces with fluids. It was constructed by Cygan et. al. [8] in order to create a general force field, which means it is capable of modelling a multitude of different systems. It contains two sets of potentials, an unbonded two-body potential, and a bonded potential which consists of a two-body and three-body term. The reason for the two discrete parts of the force field, is the fact that the bonds in the ClayFF model are static. When the system is initialized, bonds are created between particles, and these are kept throughout the whole simulation. It is therefore not capable of simulating chemical reactions, but it is nevertheless a rigorous model for the mechanical dynamics of atomic interactions.

The unbonded potential is determined by the electrostatic interactions, and van der Waals forces based on the Lennard-Jones model:

$$V_{unbonded} = V_{Coulomb} + V_{vdW} = \frac{e^2}{4\pi\epsilon_0}\sum_{i\neq j}\frac{Z_i Z_j}{r_{ij}} + \sum_{i\neq j} D_{ij}\left[\left(\frac{R_0}{r_{ij}}\right)^{12} - \left(\frac{R_0}{r_{ij}}\right)^{6}\right] \tag{5.9}$$

The first term is the well known Coulomb potential, with parameters e as the electron charge, ϵ_0 is the dielectric constant and Z_i is the partial charge of the atom. The second term is the Lennard-Jones potential, with parameters as described in the previous section. These parameters are often only known for a monoatomic system. In the ClayFF potential, the Lorentz-Berthelot mixing rules are usually applied when a pair of atoms of different species are interacting. This mixing rule states that the Lennard-Jones parameters of two different particles are approximated by

$$R_0^{ij} = \frac{1}{2}(R_0^i + R_0^j) \tag{5.10a}$$

$$D_{ij} = \sqrt{D_i D_j} \tag{5.10b}$$

The number of parameters will be a linear function of number of particle species in a simulation, instead of a quadratic function. This will therefore significantly reduce the required number of parameters needed to perform a simulation.

The bonded part of the potential is modeled by a harmonic oscillator. The two-body and three-body terms of this potential are given as:

$$V_{ij} = k_r(r_{ij} - r_0)^2 \tag{5.11a}$$

$$V_{ijk} = k_\theta(\theta_{ijk} - \theta_0)^2 \tag{5.11b}$$

k_r and k_θ are the strength of the bonded and angular potential respectively. r_0 and θ_0 are the equilibrium length and angle.

The parameters used in our experiments can be found in [8], the SPC model was chosen for the parameters of water.

5.2.1 Damping of long-range potentials

In comparison to the LJ-potential the Coulomb-potential does not fall towards zero fast enough to use a cutoff. Gauss' law states that the electrical flux through a closed surface is proportional to the total charge inside of the surface. The long range effects of the coulomb potential can therefore not easily be discarded. Methods like Ewald summation deals with this problem. We are however going to try to approximate the Coulomb interaction using a short range potential.

By assuming a charge neutral system, it was shown by Carré et. al. [9] that the Yukawa potential (5.12) is a good approximation to the Coulomb potential when introducing a cutoff radius. It may however cause some unphysical behaviour when modeling water. These unphysical bahaviours are hovewer not the main source of errors in the simulations performed in this thesis, and it will therefore be a sufficient approximation. We will use the USC force field when a more rigorous model is required.

$$V_{Coulomb} = \frac{1}{r} \rightarrow V_Y = \frac{D_Y e^{-r/r_s}}{r} \tag{5.12}$$

Here r_s is a screening parameter, which damps the potential. D_Y is a scaling parameter which helps the Yukawa potential be a better fit to the Coulombs potential when introducing a cutoff and screening length.

5.3 Vashishta-Kalia-Rino-Ebbsjø potential

The Vashishta-Kalia-Rino-Ebbsjø potential [10], henceforth abbreviated USC(University of Southern California) from where it was developed, is a molecular dynamics potential originally used to study crystalline and amorphous structures of silica. It was later expanded to model silica-water interfaces as well. In contrast to the ClayFF model, it is a reactive force field. Therefore it is capable of modelling simple chemical reactions, like the formation of silanol groups in silica-water interfaces.

The potential form itself is composed of a two body and three body part. The two body interaction is determined by four terms: the steric repulsion, Coulomb interactions, dipole interactions and van der Waals attractions.

$$V_{ij}(r) = \frac{H_{ij}}{r^{\eta_{ij}}} + \frac{Z_i Z_j}{r} e^{-r/r_{1s}} - \frac{D_{ij}}{2r^4} e^{-r/r_{4s}} - \frac{W_{ij}}{r^6} \tag{5.13}$$

H_{ij} and η_{ij} are the steric repulsion parameters. Z_i is an effective charge of an atom. D_{ij} is a charge-dipole parameter, and W_{ij} is the strength of the van der Waals interaction. In addition, the Coulomb and dipole interactions have associated screening lengths r_{1s} and r_{4s}.

A three body term models the bending of covalent bonds. It takes the form

$$V_{ijk}(\mathbf{r}_{ij}, \mathbf{r}_{ik}) = B_{ijk}\left(\cos\theta_{ijk} - \cos\theta_0\right)^2 \exp\left(\frac{\xi}{r_{ij} - r_0} + \frac{\xi}{r_{ij} - r_0}\right) \tag{5.14}$$

Unlike ClayFF, this three body term is present for the interactions between oxygen and silicon atoms, as well as between oxygen and hydrogen.

Parameters used by this potential model can be found in [11] and [10].

5.3.1 Bond order scheme

The USC-potential models simple chemical reactions, and contains two states for the oxygen(the silica, and water states). As the two states produce different behaviour of the O-O interaction, a method to distinguish the two are needed. In the ClayFF model this was done by representing the oxygens in silica and water as two distinct particles. This is however not possible in the USC-model due to the introduction of bond breaking and formations. Another method was proposed which distincts the two states by interpolating the potential between them. The resulting O-O interatomic potential is modelled as equation (5.15).

$$V_{ij} = f_{ij}V_{ij}^{silica} + (1 - f_{ij})V_{ij}^{water} \tag{5.15}$$

The interpolation parameter f_{ij} is determined by the surroundings of the two oxygen atoms i and j. It is the weighted average, given by equation (5.16), of the number of hydrogen atoms compared to silicon atoms surrounding the two oxygen atoms.

$$f_{ij} = \frac{n_i^H + n_j^H}{n_i^H + n_j^H + n_i^{Si} + n_j^{Si}} \tag{5.16}$$

where n_i^β is the number of β-particles(β being hydrogen or silicon) surrounding oxygen number i. In order to have a smooth transition between the silica and water states, the number of β particles are determined by a Θ function as

$$n_i^\beta = \sum_{k\in\beta} \Theta_\beta(r_{ik}) \tag{5.17}$$

where Θ is a function which varies smoothly from 0 to 1.

$$\Theta_\beta(r) = 1 - \frac{r - R_\beta + D_\beta}{2D_\beta} + \frac{\sin(\pi(r - R_\beta + D_\beta)/D_\beta)}{2\pi} \tag{5.18}$$

5.3.2 Implementation details

One of the major new points is the introduction of dynamic bonds, through the three body potential. When neighbour lists are constructed, which is discussed in section 2.2, a second list is constructed which contains particles within the range of the three body force field. Also if the atom is an oxygen, the number of silicon and hydrogen atoms surrounding the oxygen are counted in this step.

The introduction of the oxygen state interpolation procedure by the bond order scheme, introduces the f_{ij} parameter which is dependant on the position of all non-oxygen neighbours. This parameter leads to some complications when determining the forces between two oxygen atoms, as shown by the expression 5.19.

$$\begin{aligned}\mathbf{F}_{O-O} &= -\nabla V_{O-O} = -\nabla(V_{silica}(1-f) + V_{water}f) \\ &= (f-1)\nabla V_{silica} - f\nabla V_{water} + \nabla f(V_{silica} - V_{water})\end{aligned} \tag{5.19}$$

The ∇f factor is dependant upon the positions of the two oxygen atoms, as well as all of the surrounding non-oxygen particles. This introduces a three body interaction due to the smooth nature of the $\Theta(r)$ function. We are however going to assume this term is vannishingly small compared to the other 2 body term. The $\Theta(r)$ function is just varying inside a small interval $R - D < r < R + D$ and can therefore be neglected ($\nabla f \approx 0$).

5.4 Potential tabulation

Calculation of forces are the most computationally costly procedure in the simulation, and is therefore critical to make this procedure as efficient as possible in order to optimize the program. There are two parts in the particle interaction calculation. One is finding particle pairs which interact, where an efficient approach to this problem is described in section 2.3. The other is the calculation of force and potential values for an interactive particle pair.

The evaluation of the exponential function is a computationally costly operation. This is present in both potential models, via the screening Coulomb and dipole interactions in USC, and the Yukawa potential term in ClayFF. Removing these computations will therefore greatly speed up the integration procedure. This problem is solved by precalculating the unbonded two-body forces between two particles at given distance intervals, and store the values in an array. For an array size of N_a, the values of the potential will be precalculated at the positions r_n, given as follows.

$$r_n^2 = \frac{n r_c^2}{N_a - 1}, \quad 0 \leq n < N_a \tag{5.20}$$

where n is the array index. By using this type of tabulation points, the forces can be read from the array when the squared distance between two particles is known. This eliminates the square root evaluation needed to compute the distance, which would have been a time consuming operation.

The two body potentials are function of the distance between particles. Therefore the forces due to this potential will be expressed as

$$\mathbf{F}_{ij} = -\nabla V(r_{ij}) = -\frac{\mathrm{d}V(r_{ij})}{\mathrm{d}r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.21}$$

It will therefore be memory efficient to store $|\mathbf{F}_{ij}|/r_{ij}$ instead of every component of $\mathbf{F}_{ij}$. When the forces are needed on component form, we simply multiply the stored value by the required component of $\mathbf{r}_{ij}$.

When we want to retrieve the value between two tabulated points, the stored value of the two closest points are linearly interpolated. This is outlined in Listing 5.1.

```cpp
void tabulateTwoBodyPotential(int type1, type2){
    double r;
    for(int n = 0; n < Na; n++){
        r = sqrt(n/dr2); //dr2 = rc^2/(Na -1)
        potentialTable[type1][type2][n] = twoBodyPotential(r, type1, type2);
        forceTable[type1][type2][n] = twoBodyScalarForce(r, type1, type2)/r;
    }
}

double twoBodyForce(Particle* p, Particle * q){
    double nFloat, f, V, Fv, t;
    int n;
    int type1 = p->type;
    int type2 = q->type;
    double r2 = 0;
    //Finding squared distance
    for(int d = 0; d < 3; d++){
        rv[d] = q->r[d] - p->r[d];
        r2 += rv[d]*rv[d];
    }
    // Interpolation procedure
    nFloat = dr2i*r2; //dr2i = 1/dr2
    n = (int)nFloat;
    t = nFloat - n;
    f = (1 - t)*forceTable[type1][type2][n] + t*forceTable[type1][type2][n + 1];
    V = (1 - t)*potentialTable[type1][type2][n]
        + t*potentialTable[type1][type2][n + 1];
    //Returning forces to component form
    for(int d = 0; d < 3; d++){
        Fv = f*rv[d];
        p->F[d] += -Fv;
        q->F[d] += Fv;
    }
    return V;
}
```

Listing 5.1: A C++ code snippet which implements the tabulation, and interpolation of the two–body forces and potentials. The tabulated force and potential values are stored in an $N_t \times N_t \times N_a$ array, where N_t is number of unique atomic species.

Chapter 6

Molecular structures

The duration of a simulation is limited, and is on the order of nanoseconds for large systems. In order to avoid simulating for long periods of time to reach a desired temperature, an initial state which resembles the desired state well is desired. This is accomplished by using an initial state which exibits some of the properties of the desired system. Methods for initializing silica, amorph silica and bulk water systems are presented in the next sections.

In order to start of with a temperature as close to the desired one as possible, the particle velocities will be given by the Boltzmann distribution. Each component is chosen from a normal distribution with standard deviation $\sqrt{k_B T/m}$, where m is the mass of the particle and T is the desired temperature. The total momentum is subtracted, after distributing velocities, in order to avoid drifts.

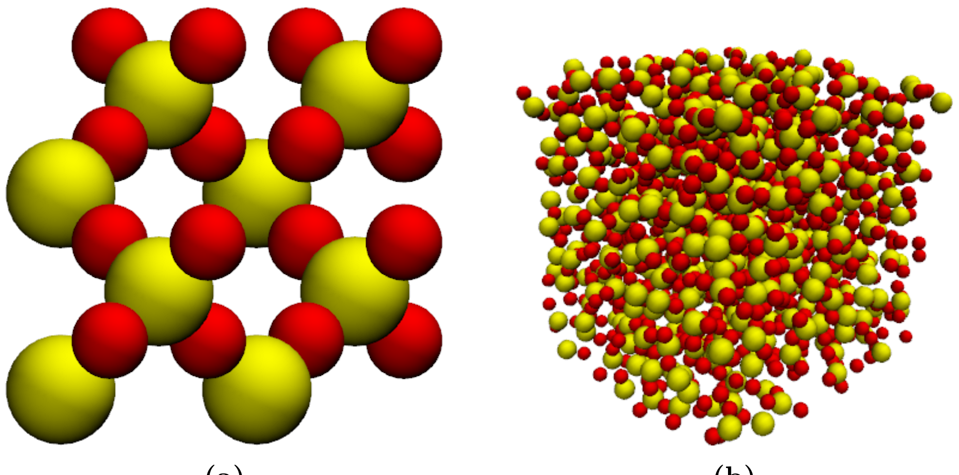


Figure 6.1: A top-down view of a cristobalite unit cell is shown in (a). An amorphous state of silica, after the melting and cooling procedure, is shown in (b).

The initial temperature will not be stable, due to the interatomic interactions.

A simulation using a thermostat will therefore be needed in order to reach a state with a stable desired temperature. This will be performed in two steps. First the Berendsen thermostat is applied to reach a desired temperature, then a simulation using the Nosé-Hoove thermostat is performed in order to stabilize the system at this temperature.

6.1 Silica

There exist several different types of structures for silica, examples include different types of quartz and cristobalite. The initial structure which is of most importance for simulations of this thesis, is β-cristobalite which is a crystalline form of silica. A unit cell of β-cristobalite is pictured in Figure 6.1a.

β-cristobalite consist of tetrahedral structures where a silicon atom is surrounded by four oxygen atoms. The unit cell structure is made up of four of these tetrahedral structures, and four additional silicon atoms which binds nearby tetrahedral structures together. β-cristobalite has a density of 2.2 g/cm^3, which gives a unit cell size of 7.16 Å.

By heating and quenching β-cristobalite, the atoms are shuffled and will transform into a silicate glass(Figure 6.1b), also called amorphous silica. In this form, the silica retains its tetrahedral structure, but becomes non-homogenous and therefore has no long-range correlations.

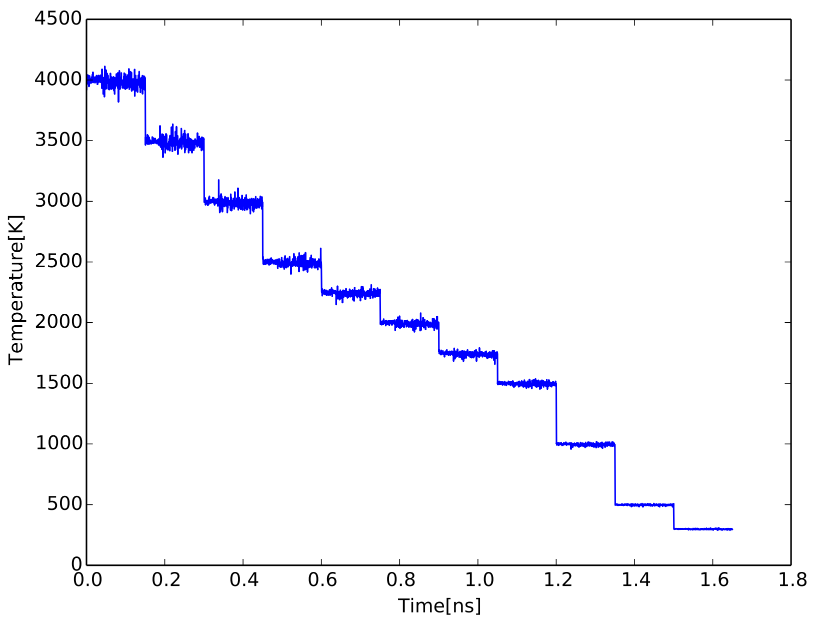


Figure 6.2: Temperature plot of the annealing procedure used to create amorphous silica.

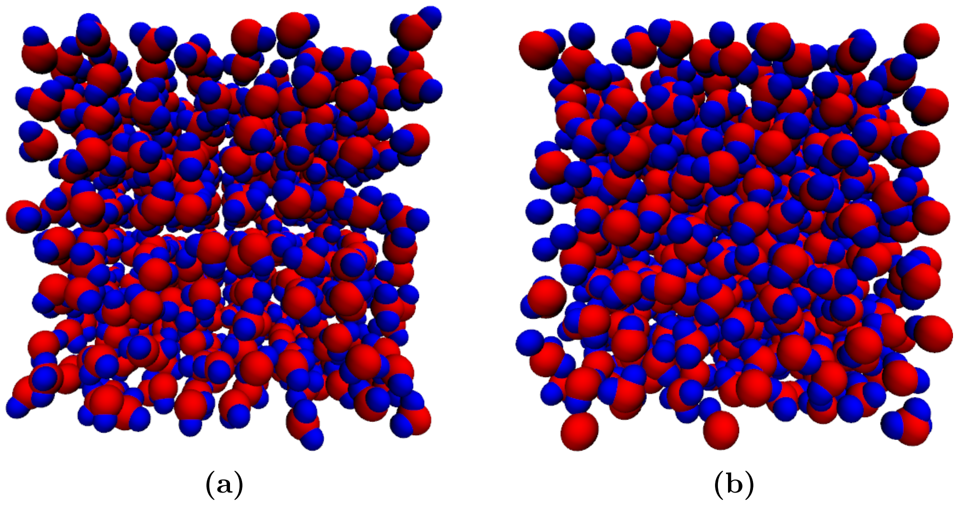


Figure 6.3: A small system of newly created bulk water is shown in (a). After a thermalization procedure, state (b) is produced.

6.1.1 Creating amorphous silica

Amorphous silica is a non-crystalline form of silica. Just like β-cristobalite it is composed of tetrahedral silica, but the structure is isotropic in nature. It does contain some ordering beyond the tetrahedral structures, and is therefore very hard, or near impossible, to create procedurally.

In order to create a system of amorphous silica we therefore need to perform a simulation of the physical procedure which is able to make amorphous silica. The system is first heated to 4000K, which makes molten silica. In order to create the short range ordering observed in amorphous silica in nature, the system is cooled down in multiple steps. Steps of 500K are used, with an additional step before and after the melting point at about 2000K, and the last one from going from $500K$ to $300K$. The temperature during the cooling procedure is presented in figure 6.2.

6.2 Water

The USC and ClayFF models are not capable of simulating solid ice, and therefore water will always be a liquid in our simulations. This means that we will have greater freedom in how we choose to initialize the system, as the structure is not bound to a constant crystal lattice as with the initialization of β-cristobalite.

A system consisting of bulk water, consists of water molecules which have a brownian motion. Therefore an initial structure will after a short time be transformed into a seemingly random state of particles. The form of the initial structure should make this transition as quick as possible, by incorporating the randomness which is observed in bulk water. The easiest way to perform this, is to assign a pseudo-random position to each particle, and a velocity chosen by the Boltzmann distribution.

The simplest way to initialize water is to partition the simulation domain into $N_x \times N_y \times N_z$ cells, and place one water molecule inside each cell. In order to decrease the uniformity, the particles are rotated by a random amount and displaced randomly inside of its cell. By performing the displacement while making sure there is at least a distance d between neighbouring molecules, no water molecules will overlap.

An initial state and a state after a short simulation of bulk water is presented in figure 6.3

Chapter 7

Numerical simulations

In this section we are going to describe several numerical experiments which was performed using one or both of the implemented force fields. The setup of the various systems, and the simulation procedure is described, followed by results and data.

We begin by looking at how efficient our implementation is, and how well it scales when number of processors are varied. Thereafter we perform several experiments, which tries to verify the code by comparing with experimental data. This includes finding the melting point of silica, looking at bond lengths and valence angles using the radial and angular distributions, and lastly measuring the density of silanol groups on a saturated silica surface. An experiment involving the contact properties of different structures of silica is also presented.

All simulations performed in the NVT ensemble are performed by the use of the Nosé-Hoover thermostat with a chain length of 3, unless otherwise stated. Periodic boundary conditions are always applied, and the temperature is always set to 300 K unless specified otherwise.

7.1 Efficiency scaling

We have distributed the total workload on a number of processors. One would perhaps guess that the time it takes to run a simulation is inversely proportional to the number of processors used. This is however not the case, as further complexity is introduced when distributing the workload between more processors. This complexity includes overhead caused by the communication of information between processors. It also includes the increased interparticle calculations required due to the increased ratio of particles on the haloes to total particles in the system. In order to draw conclusions of what the most efficient domain partitioning for a system is, we need a way to measure the efficiency scaling. This is performed by the two different methods described below.

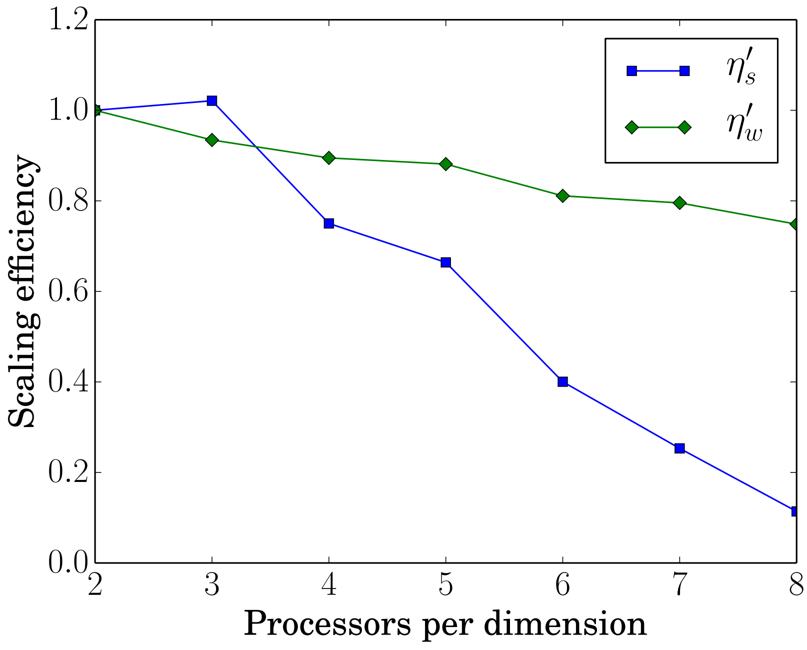


Figure 7.1: The two efficiency scaling measurements are plotted as a function of number of processors in each dimension. This means the number on the x-axis should be cubed in order to get total number of processors used for the given data point.

7.1.1 Strong scaling

The strong scaling efficiency is a measure which determines how well the algorithm scales when the simulation domain and particles are held fixed, while increasing number of processors. It is defined as

$$\eta_s = \frac{t_1}{N t_N} \tag{7.1}$$

where t_1 is the run time while using one processor, and t_N is the run time while using N processors.

There is a limitation to our implementation of the molecular simulation. Particles on the halo are stored the same way as internal particles: in an array. This limits the number of ones particle image in a processor to one. The consequence is that the system size has to be at least $4 \times 4 \times 4$ cells, while using 2 processors in each dimension, of a total of 8. Due to this limitation, it is not possible to run the code using a single processor. The strong scaling is therfore redefined as:

$$\eta'_s = \frac{8 t_8}{N t_N} \tag{7.2}$$

By inserting $N = 8$ in equation (7.1) it is easy to show that the new scaling is proportional to the original strong scaling: $\eta'_s(N) = \eta_s(N)/\eta_s(8)$. The interesting part of the scaling function is its shape as function of number of processors. Therefore this will not change the conclusion drawn from the data, even if we have introduced this unknown constant.

In this benchmarking, a system consisting of $10 \times 10 \times 10$ unit cells of SiO_2 was used, which results in 24000 atoms. The average time used for 15 simulations were used for each data point.

N_{CPU}	N_{atoms}/N_{CPU}	Time[s]	η'_s
8	3000	3080	1.00
27	889	894	1.02
64	375	513	0.75
125	192	297	0.66
216	111	285	0.40
343	70	284	0.25
512	47	422	0.11

Table 7.1: Benchmarking results of the strong scaling η'_s.

The reason for the badly scaled efficiency when increasing processors, is the fact that the system size was chosen based on the time it would take to simulate it using 8 processors. This resulted in very small local systems for higher number of processors, thereby causing a lot of overhead. In order to investigate the strong scaling further, a much larger system should be used.

7.1.2 Weak scaling

The weak scaling efficiency is a measure of how well the program efficiency scales when the system size is proportional to the number of processors. It is measured by keeping the number of particles and volume per processor constant, while varying number of processors. It is defined as:

$$\eta_w = \frac{t_1}{t_N} \tag{7.3}$$

However, just as we did with the strong scaling, we need to modify this expression due to the limitations in the program. The new weak scaling takes the form

$$\eta'_w = \frac{t_8}{t_N} \tag{7.4}$$

And just as with the strong scaling, the redefined weak scaling efficiency is a scaled version of the original: $\eta'_w = \eta_w(N)/\eta_w(8)$.

In this benchmarking, each processor is assigned $3 \times 3 \times 3$ unit cells of SiO_2. This is simulated in the microcanonical ensemble(NVE) for a duration of 100 ps.

N_{CPU}	N_{atoms}	Time[s]	η'_w
8	5184	783	1.00
27	17496	838	0.93
64	41472	876	0.89
125	81000	889	0.88
216	139968	966	0.81
343	222264	985	0.80
512	331776	1047	0.75

Table 7.2: Benchmarking results of the weak scaling η'_w.

The implemented simulation procedure seems to scale very well according to the data presented in table 7.2. There is only a slight decay in efficiency. This decay must be due to communications of the local energy each timestep, which is a global communication process. This halts every processor, and may therefore introduce a bottleneck. In the case of particle communication, the processors are only interacting with neighbouring processors. This may however also introduce some overhead, as there are 16 processing units per rack at Abel. Communication between racks takes more time than communication between processors in the same rack. When the number of processors increase, the total number of messages between racks increases as well, which results in more overhead.

7.2 Code validation

There is no way around making at least a small number of errors when writing large pieces of code. Mostly it is just a typographical error. Other times the error is more severe, which needs more effort to detect and fix. Generally there are two different kinds of errors which makes the implementation not work as intended. The first one is typographical errors, or syntax errors. These are in most cases detected by the compiler, in the compilation process, or they result in segmentation faults when the program is run. The other kind of faults are due to the implemented algorithm. These errors result in perfectly runnable code, but the results are in most cases quite different from what is expected. For instance, the author experienced errors in the form of particles not forming bonds when the USC model was applied. The reason for this odd behaviour was the fact that some parameters suplemented by the article which described the method [11], had listed some parameters as a scaling of an atomic unit, whereas other parameters were not scaled. The correct parameters were eventually found by determining them from the simulation code supplemented by the article.

It is therefore of utter importance to test code. In the next following sections this is performed by comparing macroscopic observables, which are sampled during a simulation, to the physical known values.

7.2.1 Radial distribution

The radial distribution function is a measurable quantity which describes the local structure of a medium. It is given as the function $g(r)$ and describes the probability of finding a particle within a given range r of another particle, relative that of an ideal gas. As it can be measured by experiments such as neutron or x-ray scattering, it is a quantity which can help us compare the results from simulations using a force field model, to the physical behaviour of the system. Therefore it gives us a glimpse of how well the force field model represents the physical behaviour.

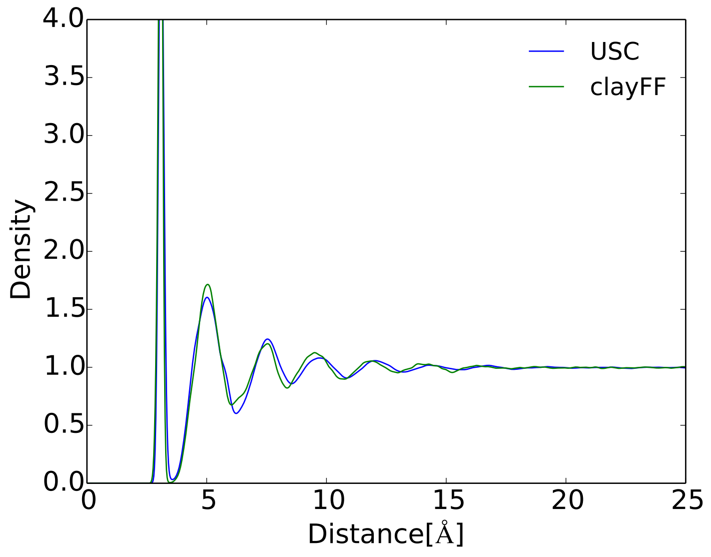
Numerically it is found by introducing distance bins, with bin size δr. The number of particles N_i located within bin number i is counted. The bin in which a particle is located, is given by $i = \lfloor r_{nm}/\delta r \rfloor$, where r_{nm} is the distance between particles n and m. This is performed by every particle pair, while using the minimum image convention. If i is larger than the number of bins, then the pair is discarded. In most cases the radial distribution of one particle species α to another β is wanted. We will define this as $g^{\alpha\beta}(r)$. If M_β is the number of particles of species β, the average number of particle pairs per volume, between two species of particles, will be given as

$$\rho^{\alpha\beta} = \begin{cases} M_\alpha M_\beta/(L_x L_y L_z), & \text{if } \alpha \neq \beta \\ (M_\alpha - 1)M_\beta/(L_x L_y L_z), & \text{if } \alpha = \beta \end{cases} \tag{7.5}$$

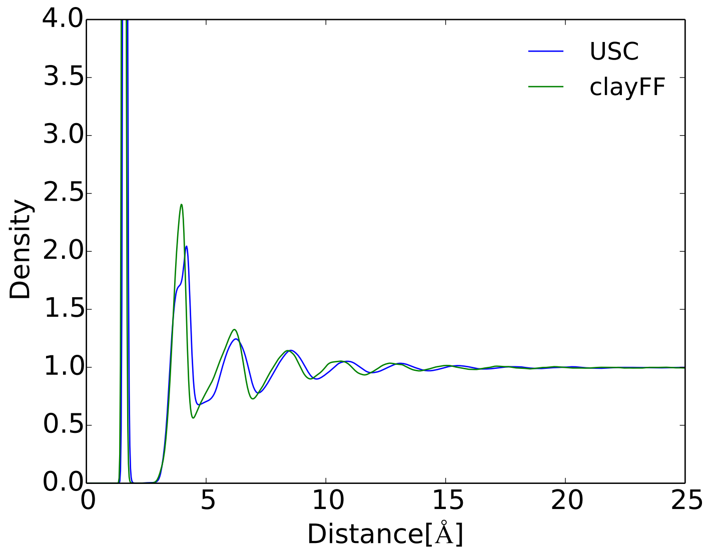
The number of particles in each bin must be scaled by this quantity in order to find $g^{\alpha\beta}(r)$, in addition to the volume of the spherical shell which represents the physical space of the bin. The expression for the radial distribution function will therefore be given as

$$g^{\alpha\beta}(r_i) = \frac{3N_i^{\alpha\beta}}{4\pi[(r_i + \delta r)^3 - r_i^3]\rho^{\alpha\beta}} \tag{7.6}$$

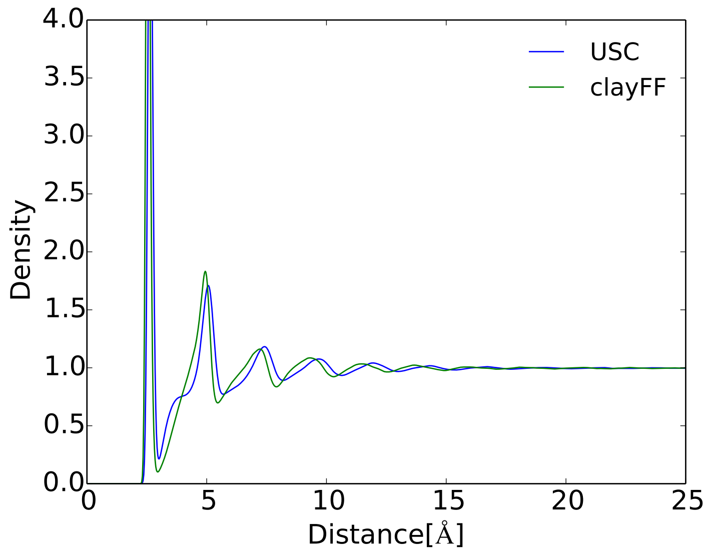
In listing 7.1, a code snippet which calculates $g(r)$ is presented.



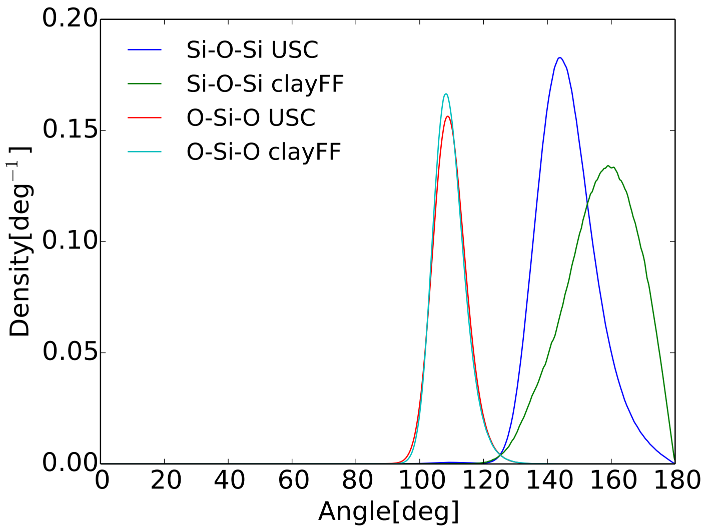
(a) Si–Si



(b) Si–O

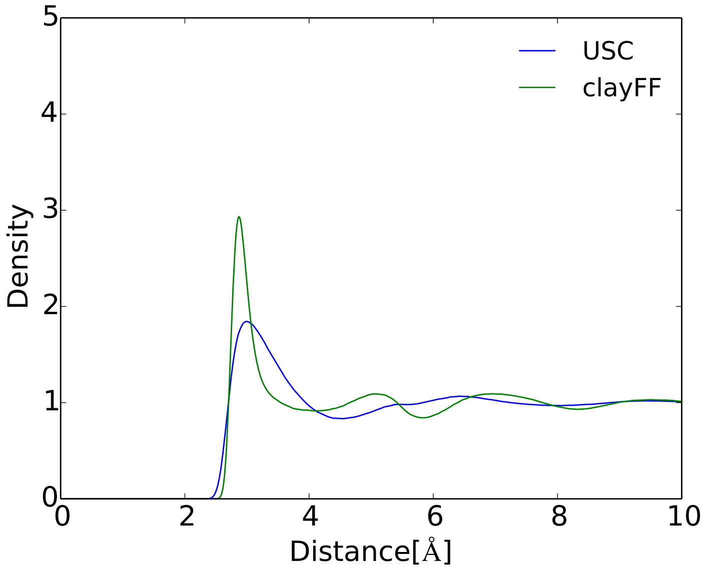


(c) O–O

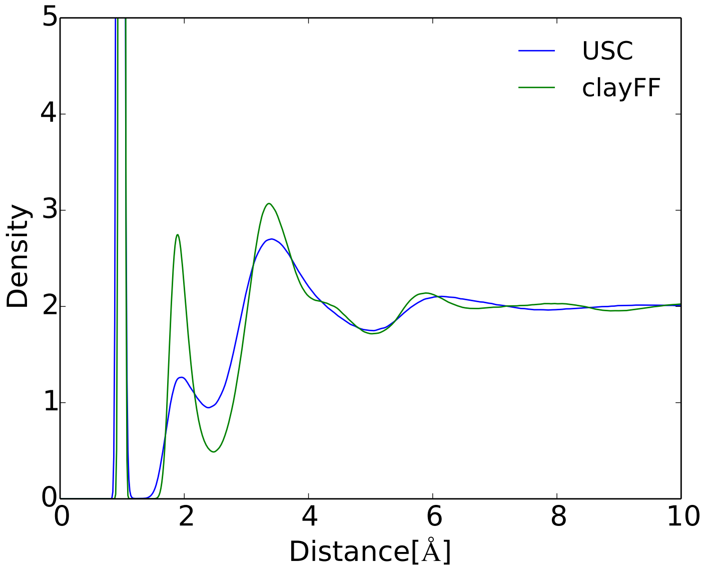


(d)

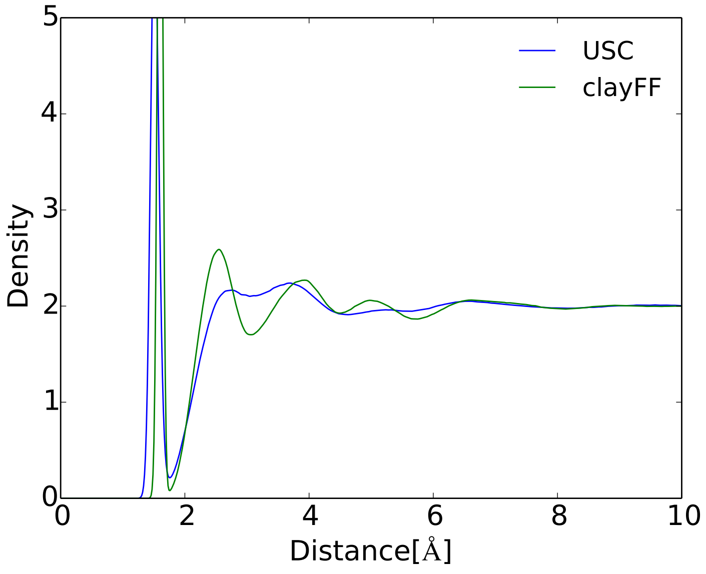
Figure 7.2: The radial and angular distribution functions of silica. Data is measured for a amorphous state of silica which is created by a heat and quenching procedure.



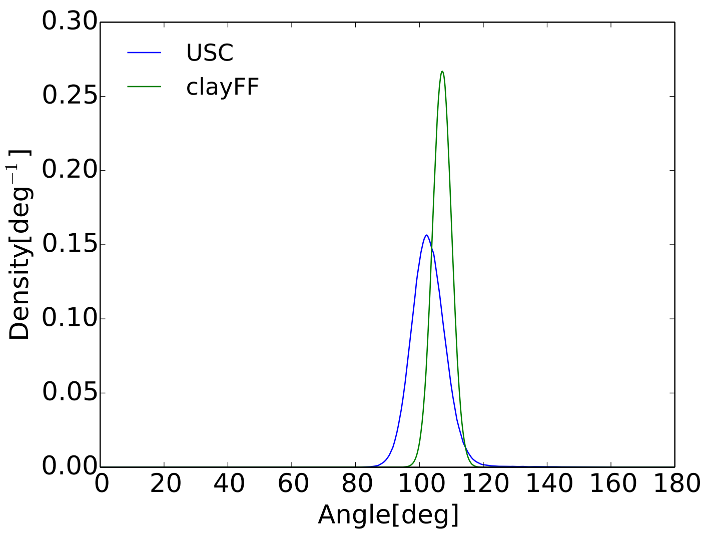
(a) O–O



(b) O–H



(c) H–H



(d)

Figure 7.3: The radial and angular distributions of water. Measurements are made after a thermalization process.

By definition $g(r) = 1$ for a ideal gas. Any deviation from this value is caused by the interatomic interactions of the medium studied. Therefore we may find indications of physical properties by studying the shape of the radial distribution. Such properties include repulsive interaction depicted by small values of $g(r)$, which means it is very unlikely to find an atom within that range. Bonds can be found by looking at the local maximum points, as the probability of finding another atom within this range increases. By this fact we may extract the average bond length by finding the position of the first peak.

```python
def radialDistribution (r, particleType, nBins, maxDistance):
    numTypes = np.max(particleType) + 1
    numParticlesOfType = np.zeros(numTypes)
    g = np.zeros([numTypes, numTypes, nBins])
    dr = maxDistance/nBins
    for n in range(numParticles):
        numParticlesOfType[particleType[n]] += 1
        for m in range(n):
            # Function which finds length of vector,
            # while enforcing minimum image convention
            r_nm = lengthOfVector(r[n] - r[m])
            i = int(r_nm/dr)
            t1 = particleType[n]
            t2 = particleType[m]
            if(i < nBins):
                g[t1, t2, i] += 1
    for t1 in range(numTypes):
        for t2 in range(t1 + 1):
            scaling = numParticlesOfType[t1]
            if(t1 == t2):
                scaling = scaling*(scaling - 1)/(Lx*Ly*Lz)
            else:
                scaling = scaling*numParticleOfType[t2]/(Lx*Ly*Lz)
                g[t1, t2] += g[t2, t1]
            for i in range(nBins):
                dV = 4/3*pi*(i**3 - (i - 1)**3)*dr**3
                g[t1, t2, i] = g[t1, t2, i]/(dV*scalingFactor)
    return g
```

Listing 7.1: Python function which computes the radial distribution function, when knowing the positions of particles, a list of particle types, number of bins and maximum allowed distance.

The radial distribution function of silica

A system of $4 \times 4 \times 4$ unit cells of silica at a density 2.2 g/cm^3 was heated and quenched using the quenching method described in 6.1.1. The radial and angular distributions were then found by applying the before mentioned algorithm. The resulting data is displayed in figure 7.2.

The different methods produce seemingly very similar graphs, for the radial distribution between all atomic pairs. However there are a slightly more detail in the plots produced by the USC model compared to the more smoothly varying plots produced by ClayFF interactions. This is most apperent when looking at the second peak of the radial distribution of Si–O, and between the first and second peak of the distribution O–O. The reason for this difference is most likely due to the three body addition in the USC, which is not present in ClayFF.

Oxygens interacting by the USC model will tend to form valence angles which are not straight, as can be seen in figure 7.2d. A state in which these angles are present are therefore more likely to be formed during the quenching process, than if particles were to interact using the ClayFF potential.

In table 7.3 the two first peaks in the radial distribution, and the maximum value of the angle distribution is listed. All values are very similar, with the exception of the Si–O–Si angle, as discussed above.

	Si–Si		Si–O		O–O		Si–O–Si	O–Si–O
Peak	1	2	1	2	1	2		
ClayFF	3.08 Å	5.04 Å	1.63 Å	3.96 Å	2.55 Å	4.49 Å	159.4°	108.3°
USC	3.10 Å	5.01 Å	1.62 Å	4.18 Å	2.63 Å	5.07 Å	143.8°	108.8°

Table 7.3: List of positions of peaks of the radial and angular distributions of silica.

The radial distribution function of water

A system of bulk water consisting of $10 \times 10 \times 10$ unit cells, for a total size of $31 \times 31 \times 31$ Å, was thermalized. It was then simulated in the NVT ensemble for 28 ps, while sampling statistics about the radial and angular distributions. This data is presented in figure 7.3.

For the various distributions of water, there are many more differences than could be seen in the distributions for silica. Most notably is the difference in the H–O–H angle. One should perhaps believe that the USC potential, which is a more advanced model, would result in an angle more similar to the one observed by experiments. This is however not the case, as can be seen in table 7.5. The physically observed bond angle for the water molecule is 109.5°, which is closer to the one measured by the ClayFF model.

The differences depicted between the first and second peak in the O–H and H–H distribution (figure 7.3b), may be caused by the fact that USC incorporates a many body force field, which makes formation of H_3O^+ and OH^- possible. In ClayFF the particles are more concentrated while the particles interacting using the USC model are a bit more spread out, which may indicate that these formations may take place.

	O–O		O–H		H–H		H–O–H
Peak	1	2	1	2	1	2	
ClayFF	2.98 Å	5.39 Å	0.97 Å	1.93 Å	1.51 Å	2.73 Å	107.1°
USC	2.87 Å	5.05 Å	0.99 Å	1.89 Å	1.60 Å	2.55 Å	102.2°

Table 7.4: List of positions of peaks of the radial and angular distributions of water.

7.2.2 Melting point of silica

In order to further verify our implementations, we approximate the melting point of silica by using the ClayFF and USC models. The melting point of a system is found by looking at the phase transition from where particles are behaving like a solid, to where they behave like a liquid. A system behaving like a solid will contain particles which are approximately fixed in space. In a liquid, particles move around more freely, and due to the effects of Brownian motion, particles will not stay put for long.

The diffusion coefficient D, discussed in section 3.3, gives an indication of the average displacement of the particles in a system. For a solid it will be practically zero, but not for a liquid, where it will increase with temperature. The phase transition is then found by locating the temperature where D starts being non-zero.

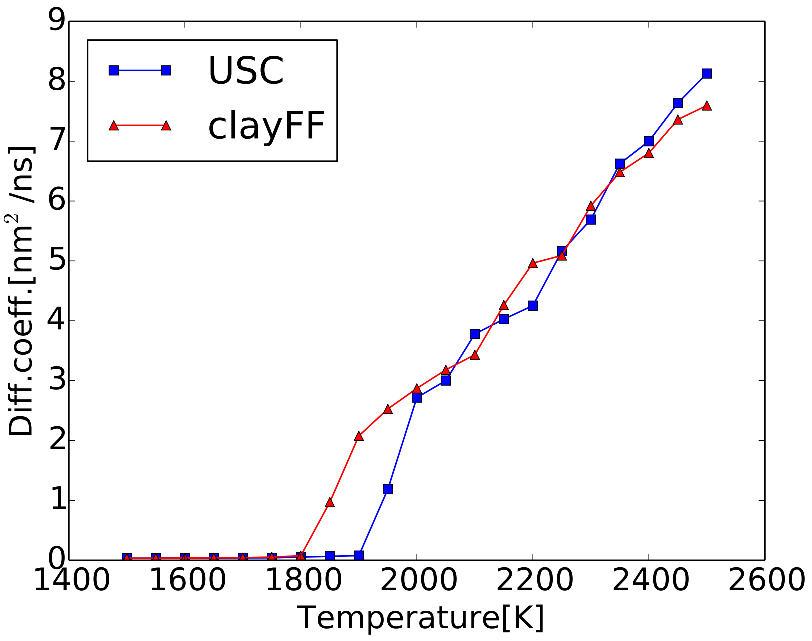


Figure 7.4: Diffusion coefficient as function of temperature during a gradual melting procedure of silica.

One side effect of simulating a system with periodic boundary conditions, is the fact that D will be linearly dependant on the inverse of the system size L [18]. This is however not of any concern when we are just trying to locate the melting point. It is nevertheless worth mentioning, as the values of D presented in the following figure will not be exactly equal to the real physical diffusion value.

The experiments were performed by using an initial system of β-cristobalite of size $49.9 \times 49.9 \times 49.9$ Å. The systems were thermalized to 1500K using the Berendsen thermostat. There were performed 21 samples of the diffusion coefficient, the first at a temperature of 1500K and the last one at 2500K. The system was simulated in the NVT ensemble by the Berendsen thermostat for 0.1 ns, with $\Delta t = 1$ fs, before a sample of the diffusion coefficient was taken. The resulting diffusion coefficients as function of temperature are presented in figure 7.4.

The data suggest the melting points of silica by using the ClayFF and USC potentials are approximately 1850K and 1950K respectively. Therefore the melting point approximated by the USC potential is closer to the true melting point of 1986K than the one measured by the ClayFF model. This difference in temperature may possibly be dependant upon the fact that the ClayFF coefficients of silica have been developed for use at room temperature. In addition, the USC potential is a more advanced model, which incorporates several additional terms in the particle interactions compared to the ClayFF potential. One should therefore expect the USC potential to be capable of simulating particles sufficiently well in a wider range of conditions than what the simpler ClayFF model is capable of.

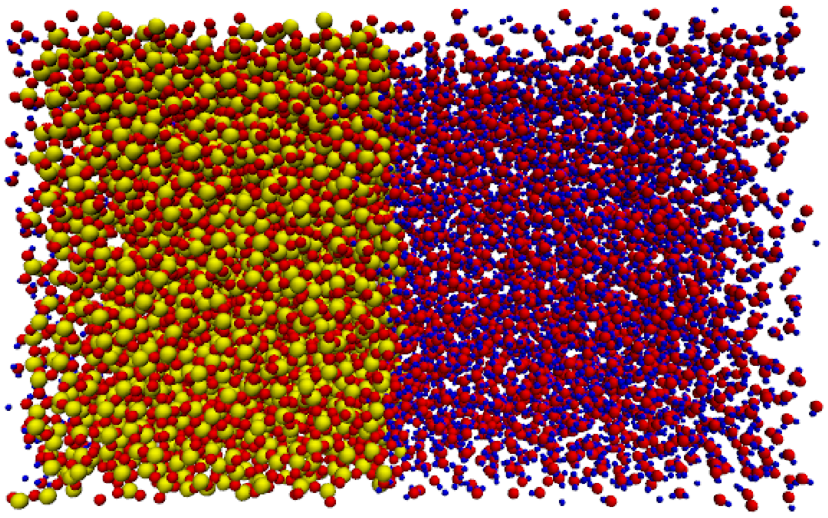


Figure 7.5: System used to measure concentration of silanol groups on amorph silica surface.

7.2.3 Silanol group density

When water interacts with the surface of silica, silanol groups may be formed. There are two ways this may happen, either an OH group is absorbed and binds with a Si atom of the silica substrate which has a coordination number less than 4. The other possible interaction is a hydrogen atom binding with an oxygen atom of the silica substrate with a coordination number of 1. These chemical reactions can be expressed as follows.

$$Si + H_2O \rightleftharpoons SiOH + H^+ \tag{7.7a}$$

$$SiO + H_2O \rightleftharpoons SiOH + OH^- \tag{7.7b}$$

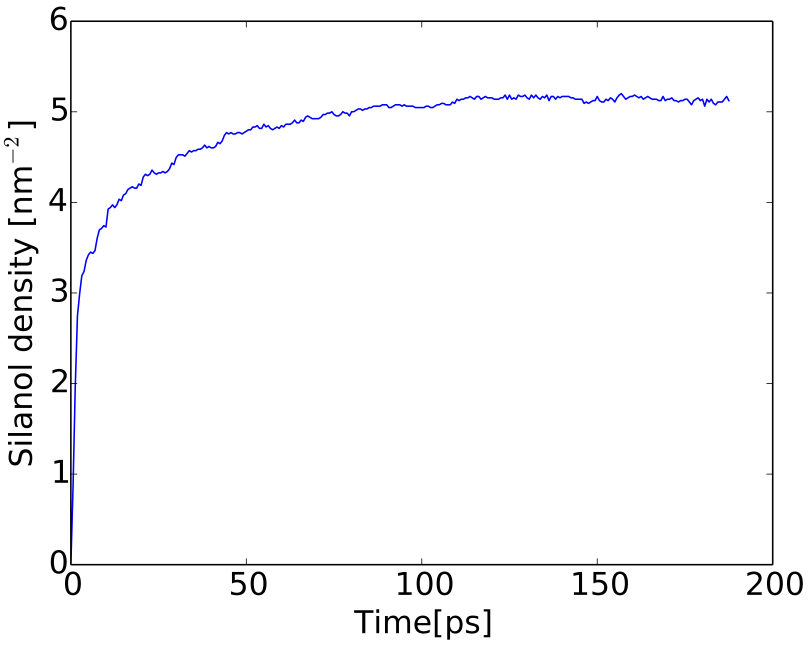


Figure 7.6: Silanol group density as a function of time on the surface of an amorph silica substrate. The data are gathered by counting the number of oxygen atoms which is within bonding range of both a silicon and hydrogen atom.

We would like to measure the silanol density on the surface of a silica substrate, and compare this with results obtained by Vashishta et. al. [11] and values obtained from experiments. Our experiment consists of a system of size $57.0 \times 57.0 \times 84.4$ Å. The silica substrate had a size of $8 \times 8 \times 6$ β-cristobalite cells, which translates to $57.0 \times 57.0 \times 42.8$ Å. The quenching procedure described in 6.1.1 was performed in order to obtain amorphous silica. There was an initial gap of 5 Å between the silica and water on both sides of the substrate. This was accounted for by a slight increase in initial density of water. The system is shown in figure 7.5.

A simulation procedure in the NVT ensemble using Berendsen thermostat for a period of 50 ps was performed in order to stabilize the system. The sampling of silanol density took place in a Noosé-Hoover simulation. The results are presented in figure 7.6.

The system is saturated after 110 ps, and converges to 5.13 nm^{-2}. This is a lower value than 5.4 nm^{-2} which Vashishta et. al. obtained. However the experimentally measured value [19] is in the range 4.9 ± 0.5 nm^{-2}, which contains the result of this experiment.

7.3 Contact properties of hydrated silica

Luan and Robbins [1] suggested that in the limit when lengths approach atomic dimensions, the classical theory of continuum contact mechanics is a badly behaved approximation. Their results indicates that the different roughness of the two objects in contact will result in different behaviour of the atoms in the contact area. In their experiments static particles were used, and measurements performed with the Lennard Jones potential model. We will therefore investigate whether the same effects will take place when a more advanced model is used for a system consisting of multiple atomic species, and whether hydrated surfaces produces different results than the dry counterparts.

The molecular system used in the experiment consists of a rigid cylindrical tip, as shown in figure 7.7, which is pushed against an eliastic block of silica. This block approximates a half-space. For half of the experiment, the cylindrical tip is surrounded by water.

7.3.1 Experimental setup

A graphical representation of the system is presented in figure 7.8. The cylinder tips have a width of 31 Å, where the two curved parts have a total width of 21 Å, and the height is 71 Å. The dimensions of the system are $113 \times 143 \times 342$ Å, where the thickness of the block of silica is 60 Å. An initial gap between the water and the silica structures were added, at 3 Å. This ensures no water molecules will overlap silica atoms and result in unphysical large forces. An increased initial density of water was added to make up for this empty space, and make sure that the density of water will stay at 1 g/cm^3 when this space is filled.

The height of the cylinder is only half of the system height. There are two reasons for this additional space. Firstly it reduces effects due to the finite size of the system. It also lets water molecules pass freely on both sides of the cylinder tip when moving it towards the silica block.

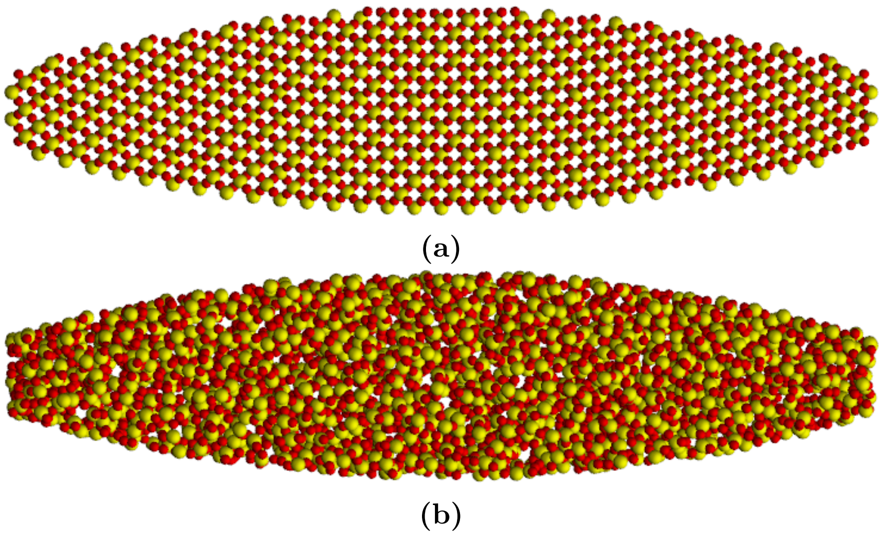


Figure 7.7: Two different silica cylinder tips. (a) has a smooth surface, whereas (b) has a more rough and non-uniform surface.

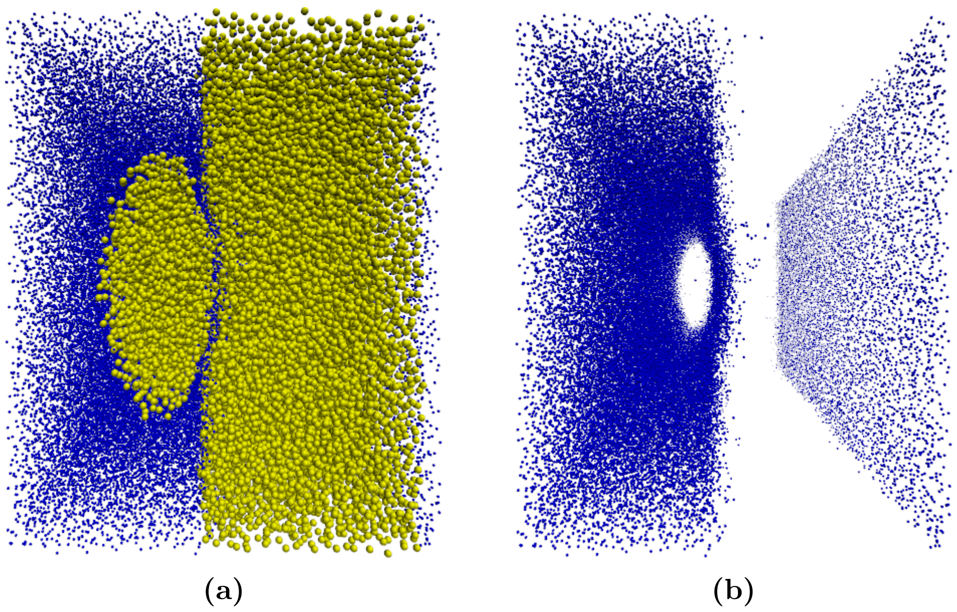


Figure 7.8: Illustrations of the pressure analysis simulation using an amorphous structure for the cylindrical tip, and with water present. In (a) the cylindrical tip is indented by 2.47 Å into the silica block. The same state is shown in (b), but with the silicon atoms removed, only showing hydrogen atoms. The oxygen atoms are not shown in any of the states, in order to better distinguish water from silica. One clearly sees from (b) that there are waetr molecules present in the contact area between the two silica substrates.

The cylinder is initially placed at a distance of 1 nm away from the silica block, this makes sure that there are no initial interactions between the two objects. In order to avoid effects due to the movement of the cylinder, it is moved at a relatively slow pace towards the block, with a constant velocity of 1 m/s. The whole procedure can be summarized in the following steps.

1. Create a rigid cylindrical tip.
2. Create silica block, which approximates a half space.
3. Fill empty space with water molecules.
4. Stabilize the system at 300 K.
5. Move the cylindrical tip towards the silica substrate with a velocity of 1 nm/s.
6. Repeat the following until the cylindrical tip is indented 5σ into the substrate:
   (a) Make measurements of potential along the y-axis.
   (b) Indent the cylinder an additional $\sigma = 2.47$ Å into the substrate.

Two different structures of cylindrical tips were studied. A bent crystalline structure, and an amorphous structure state of silica, as seen in figure 7.7. For the experiments using dry surfaces, step 3 in the above list was ignored.

7.3.2 Results

The simulations produced data of the pressure applied on the surface of the silica block, as function of y-coordinate. 6 measurements were made of the pressure for each structure, at 0 to 5 atomic silica lengths into the half-space. One atomic silica length is $\sigma = 2.47$ Å, at a density of 2.2 g/cm^3. Each measurement was sampled during a NVT simulation of 1 ns.

To extract further information from the data, a curve fitting analysis was performed in order to determine the p_0 and a parameters from equation (3.11) which best fit the data. An example of the measured pressure and its curve fitted function is shown in figure 7.9. A code snippet which performs this task is presented in listing 7.2.

The maximum pressure p_0 as a function of contact radius a is presented in figures 7.11. The pressure in the case of hydrated surfaces is smaller than for dry surfaces. This suggests hydrated surfaces are slightly more cohesive than dry surfaces of silica.

It looks like the maximum pressure behaves as a linear function of contact radius. We therefore use the continuum mechanical theory in order to extract the elastic modulus by linear regression, using equation 3.12. The results of the regression analysis are presented in the following table:

	$E^*/(2R)$	E^*
Dry amorph	0.645 eV/nm^4	8.42 eV/nm^3
Hydrated amorph	0.606 eV/nm^4	7.91 eV/nm^3
Dry bent	0.610 eV/nm^4	7.96 eV/nm^3
Hydrated bent	0.465 eV/nm^4	6.06 eV/nm^3

Table 7.5: Elasticity modulus extracted from linear regression of p_0 vs a for bent crystalline and amorphous silica substrates.

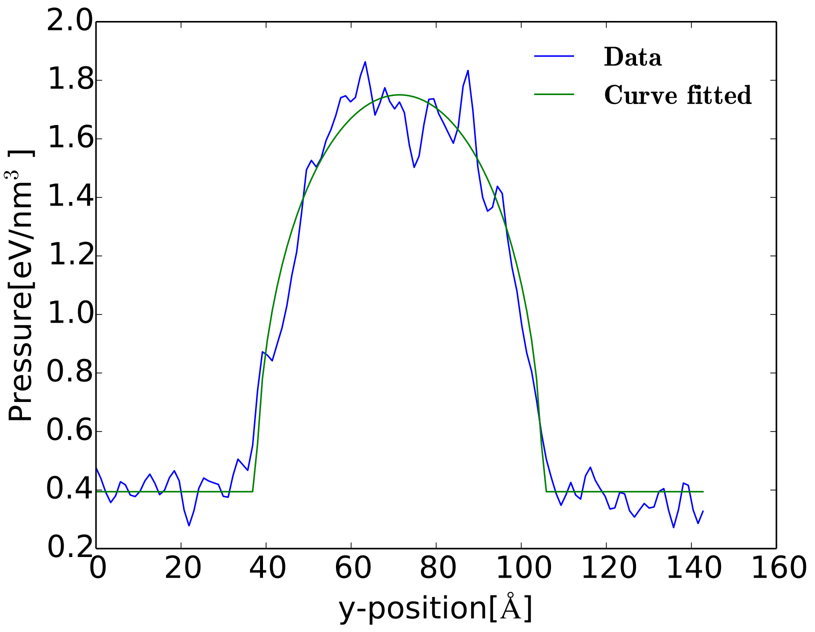
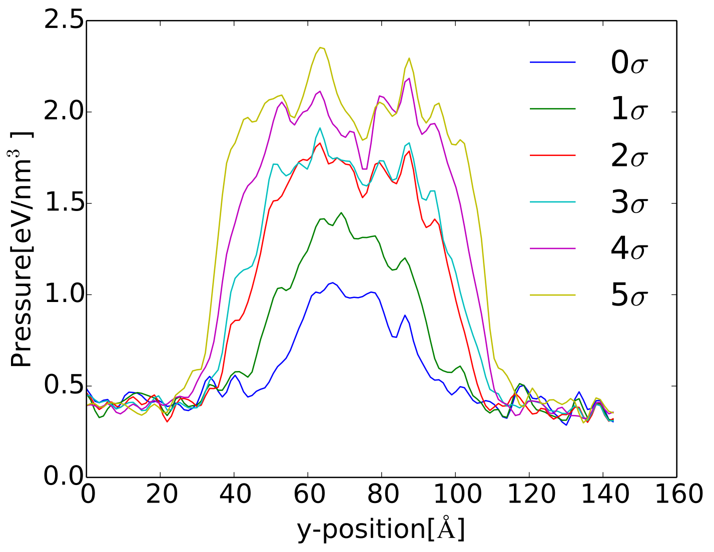
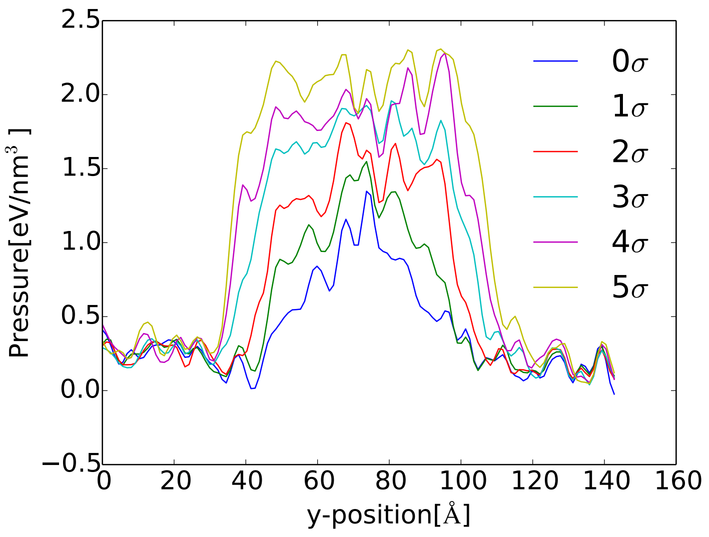


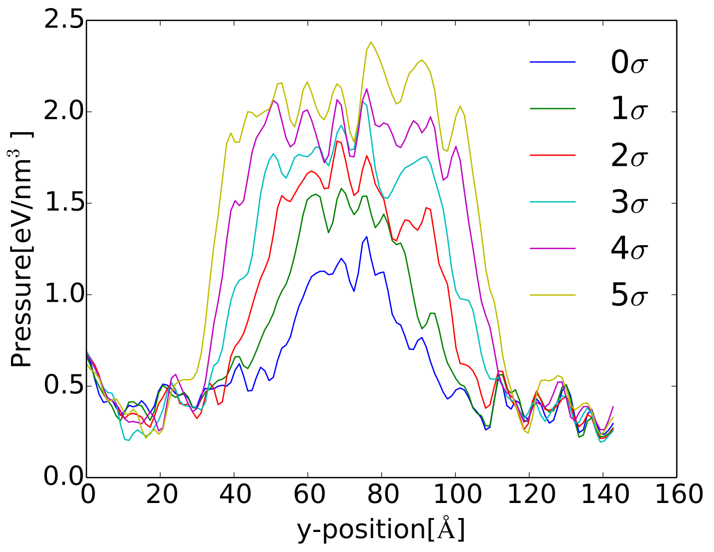
Figure 7.9: A curve fitting of pressure data. The data is produced by the amorphous cylinder tip pressed 2σ into the hydrated silica substrate.



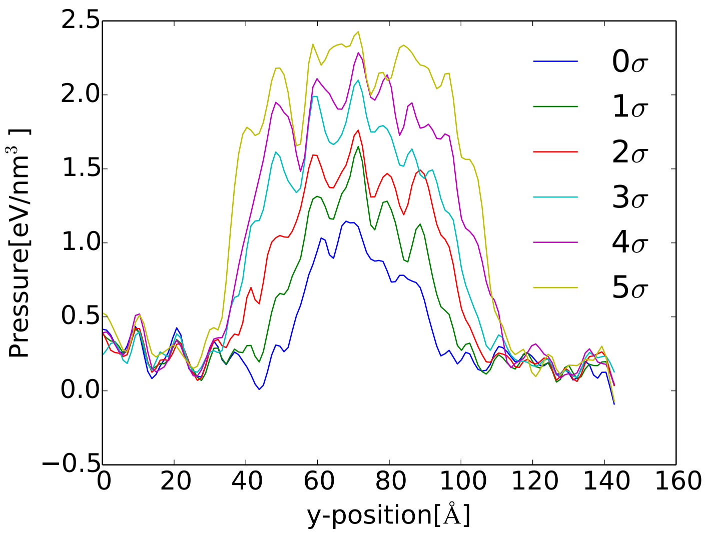
(a) Amorphous and hydrated



(b) Amorphous and dry

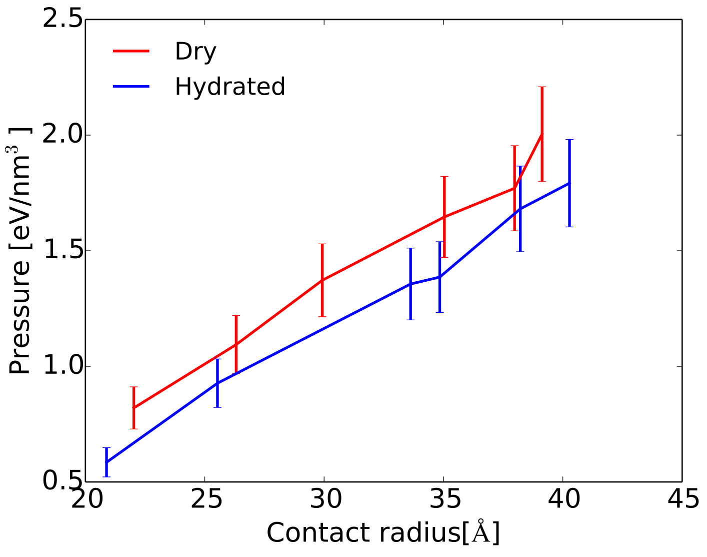


(c) Bent crystalline and hydrated

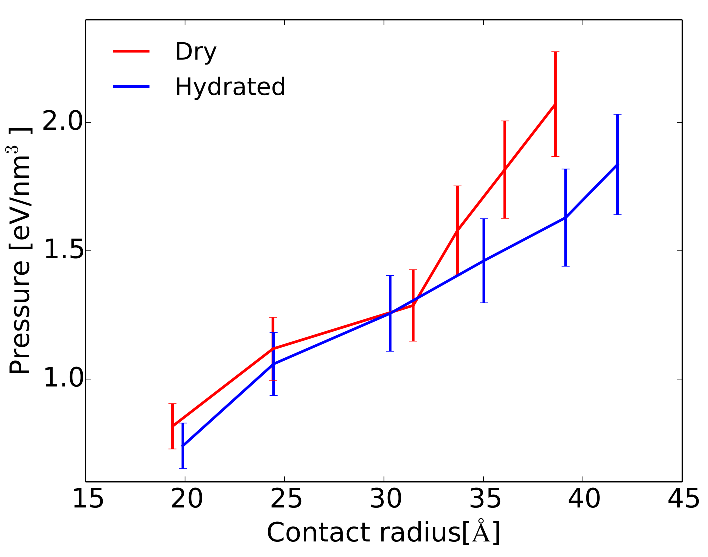


(d) Bent crystalline and dry

Figure 7.10: The pressure distribution along the y-axis at the surface of a silica substrate. A cylindrical tip is pressed against a silica substrate with indentations ranging from 0 to 5σ.



(a) Amorphous tip



(b) Bent crystalline tip

Figure 7.11: The pressure of a cylindrical tip pressed against an amorphous silica substrate as function of contact radius.

7.3.3 Notes on pressure calculation

The top layer, approximately 6 Å into the silica substrate, is used to measure the pressure from the cylindrical tip. The symmetry along the z-axis is exploited in order to efficiently calculate the average pressure as function of y-value. The cells mentioned in section 3.4 are taken to be slabs of thickness h in y-direction which span the whole system in x and z direction.

Using this simplification, the calculation of the line segment ratio for a given particle pair, l_{ij} from equation (3.9), will be nearly trivial to compute. If the two particles i and j are located in two different cells n_i and n_j respectively, the values of l_{ij} of cell n will be given as

$$l_{ij} = \begin{cases} [h(n_i+1) - y_i]/y_{ij}, & n = n_i \\ h/y_{ij}, & n_i < n < n_j \\ (y_j - hn_j)/y_{ij}, & n = n_j \end{cases} \tag{7.8}$$

Here y_i is the y-coordinate of particle i, y_{ij} is the difference in y-coordinate between particle i and j. It is assumed that $y_i < y_j$. If the two particles are located in the same cell, then $l_{ij} = 1$.

In Listing 7.3 a code snippet used to calculate the pressure is presented.

```
def linearRegression(x, y):
    #Linear regression to find A in y = A(1 -x)
    x = x/(a*a)
    A = np.sum(y*(1 -x))/np.sum((1 - x)**2)
    return A
def pressureCurveFit(x, y, N = 10000):
    xcenter = x - 0.5*x[-1]
    aa = x[-1]/2*np.arange(N)/(N -1)
    resa = 0
    resp = 0
    minerr = 10000
    #Heaviside step function
    def H(x):
        return x >= 0
    n = len(x)
    for i in range(N):
        a = aa[i]
        Hh = H(xcenter + a) -H(xcenter - a)
        nh = np.sum(Hh)
        Hy = y*Hh
        nonContactPressure = np.sum(y -Hy)/(n - nh)
        #Finding the indexis which corresponds L/2 - a and L/2 + a
        i1 = np.ceil((n - 1)*(0.5*x[-1] - a)/x[-1])
        i2 = int((n - 1)*(0.5*x[-1] + a)/x[-1])
        yc = y[i1:i2] - NonContactPressure
        xc = xcenter[i1:i2]
        p = np.sqrt( linearRegression((xc/a)**2, yc**2))
        #Finding the error by a rms comparison
        err = np.sum((yc -p/a*np.sqrt(a*a - xc**2))**2)
        err += np.sum((y[:i1] - NonContactPressure)**2)
        err += np.sum((y[i2:] - NonContactPressure)**2)
        err = err/n
        if(err < minerr):
            resa = a
            resp = p
            minerr = err
    yplt = resp*np.sqrt(1 - xcenter**2/resa**2)
    for i in range(len(y)):
        if(xcenter[i]**2 >= resa**2):
            yplt[i] = minflp
        else:
            yplt[i] += minflp
    return yplt, err, resa, resp
```

Listing 7.2: Python algorithm which performs a curve fitting of the input data by testing for different contact radii, and thereafter finding the best value of p_0 by a regression method.

```
void  virial (double y1, double y2, double w){
    int  n1 = (int)(y1/h);
    int  n2 = (int)(y2/h);
    if (n1 == n2){
        pressure [n1]  += w;
        return;
    }
    double  virialPerLineLength  = w/(y2 −y1);
    pressure [n1]  += (n1∗h + 1 −y1)∗virialPerLineLength;
    pressure [n2]  += (y2 −n2∗h)∗ virialPerLineLength ;
    w = h∗virialPerLineLength;
    for( int  i  = n1 + 1; i  < n2; i++){
        pressure [ i ]  += w;
    }
}

void  calculatePressure ( Particle  ∗p){
    double dx, dy, dz, w;
    vector<double> forces;
    //Ideal gas term
    pressure [( int )(p−>r[1]/h)]  += p−>m∗(p−>v[0]∗p−>v[0] + p−>v[1]∗p−>v[1] +
        p−>v[2]∗p−>v[2]);
    // Virial  term
    for( Particle ∗ q: p−>neighburs){
        //Avoiding to count a monitored pair twice
        if (q−>monitored && q−>id < p−>id){
            continue;
        }
        forces  = p−>forces[q−>id];
        dx = p−>r[0] − q−>r[0];
        dy = p−>r[1] − q−>r[1];
        dz = p−>r[2] − q−>r[2];
        w = forces [0]∗dx + forces [1]∗dy + forces [2]∗dz;
        //y1 < y2 is  required
        if (p−>r[1]  < q−>r[1]){
            virial (p−>r[1], q−>r[1], w);
        } else {
            virial (q−>r[1], p−>r[1], w);
        }
    }
}
```

Listing 7.3: C++ code snippet which calculates the pressure along the y-axis.

Chapter 8

Discussion and conclusion

The molecular dynamical code was aimed to be a very efficient implementation of the USC potential, which to some degree was the case. Gonnet's algorithm, described in chapter 2, sped up the total integration procedure by approximately 10%. However more efficiency improvements could have have been implemented were one to use Verlet lists in combination with the Gonnet's algorithm. With Verlet lists, a particles neighbour list would just have needed to be updated every few timesteps, instead of every single one. This would greatly increase the efficiency of neighbour list constructions, thereby decreased the time taken in order to calculate forces.

In section 7.2.1 we found the radial distribution function of silica and bulk water. For the silica system, they were similar, but with some subtle differences. The USC model produced radial distributions which had more detail, than the ones produced by the ClayFF model. Results showed that the average position of the first peak of the radial distribution, was 1.62 Å for the USC model, and 1.63 Å for the ClayFF model. These are both consistent with the experimental measured value of 1.62 Å.

There were more differences in the plots produced by analyzing the bulk water systems. This is to be expected, as ClayFF utilizes static bonds for the water model, whereas particles move more freely in the USC model. The O–H bond length was, just as with Si–O, consistent with experimental values for both models. In addition, the radial distributions of water for the USC model is identical to the data published by Vashishta et. al.[11], who also performed an analysis of the radial distribution of bulk water using the USC model.

The melting point temperature of silica was studied in section 7.2.2. This study produced slightly different results for the two models. For ClayFF the melting point was found to be approximately 1850 K, whereas for USC it ended up being approximately 1950 K. Therefore the USC model produced the most accurate results, as the real physical value is 1986 K. The data showed that it is only in the phase transition between temperatures of 1800 K to 2000 K that the diffusion coefficients are not similar between the two models. We may therefore conclude with the fact that the USC potential is a better model for phase transitions for silica, but they both model the diffusion properties of solid and molten silica equally well.

The melting point was produced by measuring the diffusion coefficient, which is known to be linearly dependant on the inverse of the system size L. In this case a system consisting of $7 \times 7 \times 7$ unit cells was used. It would be an interesting study to investigate whether varying the simulation size would affect the results. One could measure the diffusion coefficient of temperature and simulation space, and then interpolate the values in order to retrieve the diffusion coefficient of an infinitely large system as function of temperature. This investigation did unfortunately not fit in the time frame of this thesis.

Initially the study of contact properties were supposed to measure the pressure between a half space and a rigid spherical tip. This did however lead to problems, due to the amount of statistics which was possible to sample within a reasonable time frame, were very little compared to what was later gathered by the simulation using a cylindrical tip. A structure which was not investigated, is a stepped(or cut) cylindrical tip. This structure is not smooth, but it is in some sense uniform, and would therefore have been an interesting comparison to the two surfaces studied here, which was either uniform and smooth, or rough and non-uniform. Again, due to time constraints, this investigation did not fit into the the time frame of this thesis, but could be an interesting part of potential future work.

The plots presented for the pressure distributions, had fluctuations which may suggest there is some discreetness baked into the measurements. In order to achieve more accurate results, one should perhaps make measurements over longer periods of time, and also perform the experiment on a larger system. However, the results which were produced, suggested the introduction of hydration made the contact more elastic. The reduced elastic modulus for the amorphous cylinder tip was 6.1% smaller for the hydrated surface compared to the dry, and 23.9% for the bent crystalline tip. This may be an indication that hydrated silica is more cohesive than dry silica surfaces.

The difference between the pressure of the two surface structures were however minimal. The only reason this may deviate from Robbins' results is the fact that they used an ideal system, which consisted of particles that interacted using the Lennard-Jones potential. The particles investigated here, interacted by a much more advanced model, and also were not monoatomic. This may be an indication that Robbins' results may not be applicable to realistic systems, or at least not to the same degree as described in his article.

In order to investigate the dynamics of different structures of silica in more detail, one should measure other properties as well. Robbins [1] suggested that the friction, and lateral contact stiffness may vary by an order of depending on the atomic surface structure of the bodies in contact. Whereas the properties measured here, was only suggested to differ by a factor of two. Measuring friction and lateral contact stiffness would therefore possibly be more relevant when determining the difference in contact properties between the different cases studied. This would however require different measurements and methods than what was implemented in this thesis, but it may be an in interesting topic for potential future work.

Appendix A

Python framework

There are multiple parts to a molecular dynamics simulation. Firstly an initial state is needed. This may be a simple crystalline structure of SiO_2 or a more complex system like water in a silica nanopore. After this system has been created, it needs to reach a steady state. This is accomplished by applying a thermostat which forces the system to reach a desired temperature, then keeping it at this state, in order to stabilize the system. Lastly, further particle trajectories may be found in order to sample statistics about the system.

This simulation scheme consists of multiple subprocedures, and not all of them are performed by the same program. Therefore a Python framework has been developed which wraps the functionality of all the different programs into an easy to use API. In the example script in listing A.1, a simple system consisting of a $5 \times 5 \times 5$ block of β-cristobalite(SiO_2) unit cells is stabilized at $300K$, then simulated in the microcanonical ensemble for a duration of $5ps$, and finally energy and temperature is plotted and the system visualized using the VMD software.

```
import numpy as np
import os
from MDmanager import MDmanager as mdm

#Timestep length (fs)
dt = 0.25
#Number of integration steps between each time  statistics  is saved
step = 10
#Number of simulation steps
sim = 2000
#unit cell size ( results in a density of 2.2g/cm^3 of SiO2)
b = 7.131
#number of cells to simulate
c = [5, 5, 5]
#Simulation volume
s = np.array(c)*b
#creating the MD object
mdm = mdm()
```

```
#Executable binary path
mdm.setExecutable(os.path.expanduser("~/usc/bin"))
#Where to store data
mdm.setDir(os.path.expanduser("~/usc/sio2sim"))
#Configure MD properties
mdm.setTemperature(300)
mdm.setVolume(s)
mdm.setProcessors([2, 2, 2])
#Making crystal structure of sio2
mdm.makeCristobalite(c)
#Thermalizing system using Berendsen thermostat
mdm.thermalize(sim, dt, tau, 10, step)
#Stabilizing the system at the desired temperature using Nose–Hoover thermostat
mdm.noseHoover(sim, dt, tau, step)
#Simulating system while enforcing constant temperature
mdm.noseHoover(sim, dt, tau, step)
#Plotting statistics (such as energy and diffusion coefficient )
mdm.analyze()
```

Listing A.1: A simple example for how to use the Python API.

A.1 The MD object

The class which controls every aspect of the molecular dynamics simulation is called `MDmanager`, and is found in the `MDmanager.py` file. In order to use this object, the following statement should be added at the beginning of the script:
`import MDmanager; mdm = MDmanager()`
This will create an object `mdm` which is able to perform molecular simulations using the C++ molecular dynamics program.

Some of the most commonly used methods of the python API is listed here with descriptions. The first five listed methods must be called before performing a simulation of any kind.

`setExecutable(executablePath)` Set the path of the executable binary directory.

`setDir(path)` Set the path where state files are stored.

`setProcessors(p)` Set number of processors. Needs to be a list of three integers.

`setVolume(s)` Set the dimensions of the simulation. Needs to be a list of three numbers.

`setTemperature(T)` Set the temperature.

`makeWater(c, border = 1)` Fill the current simulation volume with water molecules. `c` is a list with instructions to how many molecules are created in each dimension. `border` is the minimum distance allowed between molecules. Returns a state id.

`makeCristobalite(c)` Fill the current simulation volume with a number of β-cristobalite cells in each dimension, determined by `c`. Returns a state id.

`simulate(length, dt, timesteps)` Simulate the current system in NVE, and saving `length` number of states. Each state has a difference of `timesteps` number of integration steps between each other. `dt` is the timestep length for each integration step. Returns a state id.

`thermalize(length, dt, tau, avglength, timesteps)` Simulate the current system in NVT ensemble using the Berendsen thermostat, and saving `length` number of states. Each state has a difference of `timesteps` number of integration steps between each other. `dt` is the timestep length for each integration step. `tau` is the heat bath coupling parameter, `avglength` is how many of the previous integration points are used to find the average temperature. Returns a state id.

`noseHoover(length, dt, tau, timesteps, chainLength = 3)` Simulate the current system in NVT ensemble using the Nosé-Hoover thermostat, and saving `length` number of states. Each state has a difference of `timesteps` number of integration steps between each other. `dt` is the timestep length for each integration step. `tau` is the heat bath coupling parameter, and `chainLength` is number of chains. Use `chainLength = 1` for the standard Nosé-Hoover thermostat.

`analyze()` Plots energy, temperature and mean squared displacement as function of time.

`loadSimulation(path, length = None)` Loads state number `length` of the simulation saved at `path`.

`visualize()` Visualizes the previous simulation using VMD. VMD needs to be installed and operational from the command line in order for this to work.

`radialDistribution(bins)` Calculates the radial distribution of the previous simulation, using `bins` number of bins.

`moveParticles(dr)` Moves the last state of the previous simulations particles by a the 3-array `dr`. Returns state id.

`combine(simulation1, simulation2)` Combines two states, where `simulation1` and `simulation2` are two state ids. Returns a state id.

`angularDistribution(bins)` Calculates the angular distribution of the previous simulation, using `bins` number of bins.

`makeCylinder(r0, d, h, w = 0, direction = [1, 0, 0], invert = False)` Makes a cylinder centred at `r0`, with a width of `d`, and height `h`, facing the direction `direction`. `w` is additional width which does not contribute to the curvature. If `invert = True` then a cylinder tip shape is rather removed from the system, than making one.

`selectBox(r0, r1)` Selects all particles inside the box spanned by the two vectors `r0` and `r1`. Returns a selection id.

`freeze(selection = -1, v = [0, 0, 0])` Freezes particles in place with selection id `selection` of the previous system. If `selection = -1` then the previously made selection is used. `v` is the velocity of the frozen particles.

Bibliography

[1] Luan, B. and Robbins, M.O. (2005), *The breakdown of continuum models for mechanical contacts*, Nature 435: 929-932 DOI: 10.1038/nature03700

[2] Lennard-Jones, J. E. (1924), *On the Determination of Molecular Fields*, Proc. R. Soc. Lond. 106: 463477 DOI: 10.1098/rspa.1924.0082.

[3] Klafter, J., Urbakh, M., Gourdon, D., Israelachvili, J. (2004), *The nonlinear nature of friction*, Nature 430: 525-528 DOI: 10.1038/nature02750

[4] Gonnet, P. (2007), *A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations*, J. Comput. Chem. 28: 570-573. DOI: 10.1002/jcc.20563

[5] Welling, U. and Germano, G. (2011), *Efficiency of linked cell algorithms*, Comp. Phys. Communications, 182: 611-615, DOI: 10.1016/j.cpc.2010.11.002

[6] Frenkel, D. and Smit, B (1996), *Understanding Molecular Simulation, From Algorithm to Applications* 2nd edn., Academic Press, London, ISBN: 0122673514

[7] Verlet, L. (1967), *computer experiments on classical fluids I. Thermodynamical properties of Lennard-Jones molecules*, Phys. Rev. B 159: 98, DOI: 10.1103/PhysRev.159.98

[8] Cygan, R. T, Liang, J and Kalinichev, A. G. (2004), *Molecular Models of Hydroxide, Oxyhydroxide, and Clay Phases and the Development of a General Force Field*, The Journal of Physical Chemistry 108.4: 1255–1266, DOI: 10.1021/jp0363287

[9] Antoine Carré et. al. (2007), *Amorphous silica modeled with truncated and screened Coulomb interactions: A molecular dynamics simulation study* J. Chem. Phys. 127: 114512, DOI: 10.1063/1.2777136

[10] P. Vashishta, Rajiv K. Kalia, Jos P. Rino, and Ingvar Ebbsjö (1990), *Interaction potential for SiO2: A molecular-dynamics study of structural correlations* Phys. Rev. B 41: 12197, DOI: 10.1103/PhysRevB.41.12197

[11] Wang, W., Nakano, A., Kalia, R.K., Vashishta, P., *Interatomic potentials for molecular dynamics simulations of hydrolysis and stress corrosion cracking of silica glass*, Unpublished, University of Southern California

[12] Thomas W Lion and Rosalind J Allen (2012), *Computing the local pressure in molecular dynamics simulations*, Journal of Physics: Condensed Matter 28: 284133, DOI: 10.1088/0953-8984/24/28/284133

[13] Hoover, W.G. (1985), *Canonical dynamics: Equilibrium phase-space distributions*, Phys. Rev. A 31: 1695-1697, DOI: 10.1103/PhysRevA.31.1695.

[14] Martyna, G.J., Klein, M.L, and Tuckerman, M. (1992), *Nosé-Hoover chains: The canonical ensemble via continuous dynamics* The Journal of Chemical Physics 97: 2635–2643, DOI: 10.1063/1.463940

[15] Berendsen, H.J.C. et. al. (1984), *Molecular-Dynamics with Coupling to an External Bath* Journal of Chemical Physics 81: 3684-3690, DOI: 10.1063/1.448118.

[16] Bond, S.D., Leimkuhler, B.J and Laird B.B (1999), *The Nosé-Poincaré Method for Constant Temperature Molecular Dynamics* Journal of Computational Physics 151: 114–134, DOI: 10.1006/jcph.1998.6171

[17] Martyna, G.J., Tobias, D.J and Klein M.L. (1994), *Constant Pressure Molecular Dynamics Algorithms*, J. Chem. Phys 101: 4177–4189

[18] Yeh, I.C. and Hummer, G. (2004), *System-Size Dependence of Diffusion Coefficients and Viscosities from Molecular Dynamics Simulations with Periodic Boundary Conditions*, Journal of Physical Chamistry 108: 15873-15879, DOI: 10.1021/jp0477147

[19] Zhuravlev L.T. (2000), *The surface chemistry of amorphous silica. Zhuravlev model*, Colloids and Surfaces A: Physicochemical and Engineering Aspects 173: 1–38, DOI: 10.1016/S0927-7757(00)00556-2

[20] Johnson, K.L. (1985), *Contact Mechanics*, Cambridge University Press, Cambridge, ISBN: 0521347963